UNIVERSIDAD
DE MÁLAGA

Departamento de Arquitectura de Computadores

TESIS DOCTORAL

Variable Radix Online Decimal Arithmetic

Carlos García Vega

Enero de 2017

Dirigida por:
Sonia González Navarro,
Julio Villalba Moreno

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

AUTOR: Carlos García Vega

http://orcid.org/0000-0001-9070-8165

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:
http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode
Cualquier parte de esta obra se puede reproducir sin autorización
pero con el reconocimiento y atribución de los autores.
No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): riuma.uma.es

UNIVERSIDAD DE MÁLAGA

Dr. D. Julio Villalba Moreno.
Catedrático del Departamento de Arquitectura de Computadores de la Universidad de Málaga.

Dra. Dña. Sonia González Navarro.
Profesora Contratada Doctor del Departamento de Arquitectura de Computadores de la Universidad de Málaga.

CERTIFICAN:

Que la memoria titulada "Variable Radix Online Decimal Arithmetic", ha sido realizada por D. Carlos García Vega bajo nuestra dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga y constituye la Tesis que presenta para optar al grado de Doctor en Ingeniería Informática.

Málaga, Enero de 2017

Dr. D. Julio Villalba Moreno.
Codirector de la tesis.

Dra. Dña. Sonia González Navarro.
Codirectora de la tesis.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

.....

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Índice general

Tabla de Contenidos iv

Índice de Figuras viii

Índice de Tablas x

1.- Introducción 1

1.1. Formato Decimal en Punto Flotante 2

1.2. Aritmética Online 3

1.3. Motivación de la Tesis y Contribuciones 4

1.4. Estructura de la Tesis 6

2.- Suma Decimal Online 9

2.1. Sumador Paralelo RBCD 9

2.2. Sumador DSSD 11

2.3. Sumador Decimal Online 15

2.3.1. Sumador Decimal Online (olDFA) para RBCD 15

2.3.2. Segmentación de la Arquitectura del olDFA 17

2.3.3. Stream de Datos 19

Mejora del throughput 21

2.3.4. Resultados Experimentales y Comparación 23

UNIVERSIDAD DE MÁLAGA

2.4. Conclusiones del Capítulo 24

3.- Sumador Decimal Online Multioperando y Multiformato 25

3.1. Sumador Decimal Online Multioperando 26

3.1.1. Construcción de Árboles de Sumadores basados en olDFAs 26

3.1.2. Construcción de Árboles de Sumadores basados en olDFA$_p$s 28

3.1.3. Árboles de Sumadores Segmentados 30

3.1.4. Resultados Experimentales 32

3.2. Sumador Decimal Online Multiformato 34

3.2.1. Multiformato mediante Etapa de Conversión 35

3.2.2. Multiformato mediante Diseño Específico 38

Módulo de Descomposición 40

Módulo de Selección de Formato 44

3.2.3. Segmentación del olDFA$_{Mformat}$ 46

Segmentación del olDFA$_{Mformat}$ mediante Etapa de Conversión 46

Segmentación del olDFA$_{Mformat}$ mediante Diseño Específico 46

3.2.4. Comparación entre olDFA$_{Mformat}$ mediante Etapa de Conversión y mediante Diseño Específico 47

3.2.5. Suma Decimal Online Multioperando y Multiformato . . . 48

3.2.6. Resultados Experimentales 49

3.2.7. Rendimiento del olDFA$_{Mformat}$ 50

3.2.8. Rendimiento de los Árboles de Sumadores Multiformato y Multioperando 53

3.2.9. Comparación entre Escenario 1 y Escenario 2 (S1 y S2) . . 57

3.3. Ejemplos de Arquitecturas Específicas para Cálculos Financieros . 60

3.4. Conclusiones del Capítulo 62

4.- Multiplicador Decimal Online 65

4.1. Introducción 65

UNIVERSIDAD DE MÁLAGA

4.2. Multiplicación Decimal Online 65
4.2.1. Multiplicación Decimal Online sin Especulación 68
Vector por Dígito sin Especulación 69
Sumador de 6 dígitos RBCD 70
4.2.2. Multiplicación Decimal Online con Especulación 71
Vector por Dígito con Especulación 72
Módulo Multiplicador 73
4.3. Resultados Experimentales 76
4.3.1. Especulación vs No Especulación 76
4.3.2. Comparación con Multiplicador Paralelo 77
4.4. Conclusiones del Capítulo 79

5.- División Decimal Online **81**
5.1. Introducción 81
5.2. División Decimal Online 82
5.2.1. Constantes de Selección para q_h y q_l 84
5.2.2. Módulo de Corrección, Vector por Dígito 86
5.2.3. Módulo de Suma de Residuo 87
5.3. Resultados Experimentales 87
5.4. Conclusiones del Capítulo 88

6.- Conclusiones y Trabajo Futuro **89**

Apéndices **93**

A.- Estudio de Modelos de Suma **93**
A.1. Codificaciones Redundantes 93
A.2. Modelos de Suma 96
A.3. Descomposición de v 96

UNIVERSIDAD DE MÁLAGA

A.4. Descomposición de z 97
A.5. Descomposición Válida para cada Codificación 98
A.6. Desbordamiento 103
A.7. Resultados Experimentales y Comparación 103
A.8. Conclusión 104

B.- Funciones de Conversión a RBCD **105**

C.- Descomposición para suma Multiformato mediante Diseño Específico **109**

C.1. Decomposición para X421 109
C.1.1. Descomposición Mixta (dígito RBCD + X421 con peso par) 111
C.1.2. Descomposición Mixta (dígito RBCD + X421 con peso impar) 111
C.2. Decomposición para XX21 112
C.2.1. Descomposición Mixta (dígito RBCD + XX21 ambos con peso impar) 112
C.2.2. Descomposición Mixta (dígito RBCD + XX21 uno con peso par y otro impar) 113

Bibliografía **115**

UNIVERSIDAD DE MÁLAGA

Índice de figuras

1.1. Formato Decimal en Punto Flotante (DFP). 2
2.1. Sumador RBCD Paralelo. 11
2.2. FAs con entradas de 3 posibits y 3 negabits. 13
2.3. Posibles casos de FA con posibits y negabits mezclados. 13
2.4. Descomposición de u_i y v_i. 13
2.5. Operación paralela DSSD. 14
2.6. a) Estructura para sumador paralelo DSSD. b) Online decimal full adder (olDFA). 16
2.7. Estructura del olDFA. 16
2.8. Suma de n-dígitos RBCD usando un olDFA. 17
2.9. Arquitectura olDFA Segmentada (olDFA_p). 18
2.10. Sincronización para datos con $n = 16$ dígitos para olDFA y olDFA_p. 19
2.11. Transición entre dos datos para el olDFA y el sumador paralelo. . . 20
2.12. Intervalo de iniciación (se necesita un ciclo de separación para insertar un dígito 0 extra). 21
2.13. olDFA con control de los transfers de entrada. 22
2.14. Intervalo de iniciación (sin ciclo de separación). 22
2.15. Sumadores online basados en el sumador RBCD paralelo de [31]. . 23
3.1. Dos arquitecturas basadas en olDFA para 6 operandos. 26
3.2. Dos arquitecturas basadas en olDFA_p para 6 operandos. 28

UNIVERSIDAD DE MÁLAGA

3.3. Dos arquitecturas segmentadas basadas en olDFAs para 6 operandos. 30

3.4. Dos arquitecturas segmentadas basadas en olDFA$_{\mathrm{p}}$s para 6 operandos. 30

3.5. Ciclo de reloj (retardo) de las arquitecturas Multioperando basadas en olDFA y olDFAp segmentadas y sin segmentar. 32

3.6. Área de las arquitecturas Multioperando basadas en olDFA y olDFAp segmentadas y sin segmentar. 33

3.7. Throughput de las arquitecturas Multioperando basadas en olDFA y olDFAp y segmentadas y sin segmentar. 34

3.8. olDFA$_{Mformat}$ mediante etapa de conversión. 37

3.9. El caso más simple: olDFA$_{Mformat}$ soportando dos códigos RBCD y RBCD$_{\text{-}4421}$. 38

3.10. El caso más complejo: Módulo de conversión cubriendo todos los casos de la Tabla 3.4. 38

3.11. Ejemplo de esquema de descomposición para multiformato. 39

3.12. Diagramas de bloque del olDFA y del olDFA$_{Mformat}$ mediante diseño específico. 40

3.13. Módulo de descomposición de olDFA$_{Mformat}$ mediante diseño específico. 41

3.14. Arquitectura general del olDFA$_{Mformat}$ mediante diseño específico. 44

3.15. Selección de formato para Z^1 (HW similar para las restantes señales que llegan del módulo `Mformat decomposition`). 45

3.16. Arquitectura segmentada del olDFA$_{Mformat}$ mediante etapa de conversión. 46

3.17. Arquitectura segmentada del olDFA$_{Mformat}$ mediante diseño específico. 47

3.18. Arquitectura de un árbol multioperando y multiformato. 49

3.19. Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión no segmentada). 53

UNIVERSIDAD DE MÁLAGA

3.20. Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión no segmentada). 54

3.21. Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión no segmentada). 54

3.22. Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión no segmentada). 55

3.23. Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión segmentada). 55

3.24. Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión segmentada). 56

3.25. Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión segmentada). 56

3.26. Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión segmentada). 57

3.27. Tasa de incremento (reducción) para el retardo bajo los escenarios S1 y S2 de RBCD+RBCD$_{-5421}$ SpecD y CodeConv. 58

3.28. Tasa de incremento (reducción) para el área bajo los escenarios S1 y S2 de RBCD+RBCD$_{-5421}$ SpecD y CodeConv. 58

3.29. Tasa de incremento (reducción) para el retardo bajo los escenarios S1 y S2 de los árboles basados en olDFA$_{Mformat}$s. 59

3.30. Tasa de incremento (reducción) para el área bajo los escenarios S1 y S2 de los árboles basados en olDFA$_{Mformat}$s. 59

3.31. Arquitecturas para calcular funciones financieras. 61

4.1. Algoritmo de multiplicación decimal online. 67

4.2. Arquitectura general del módulo de multiplicación vector por dígito sin especulación. 69

4.3. Arquitectura general del módulo de suma de 6 dígitos RBCD. . . . 70

UNIVERSIDAD DE MÁLAGA

4.4. Arquitectura de la multiplicación decimal online con especulación. 71

4.5. Arquitectura general del módulo de multiplicación vector por dígito con especulación. 73

4.6. Arquitectura del módulo multiplicador. 74

4.7. Arquitecturas para comparar un multiplicador paralelo con el multiplicador online propuesto. 78

5.1. Algoritmo división decimal online. 83

5.2. Arquitectura de la división decimal online. 84

A.1. Modelo 11-1. 101

A.2. Modelo 8-2. 102

A.3. Modelo 9-3. 102

UNIVERSIDAD DE MÁLAGA

Índice de Tablas

1.1. Parámetros de los formatos decimales decimal64 y decimal128 . . . 3

1.2. Dígitos RBCD 4

2.1. Tabla de verdad para la suma 3 posibits y 3 negabits 12

2.2. Tabla de verdad para sumar posibits y negabits mezclados 12

2.3. Resultados experimentales de los sumadores decimales online . . . 24

3.1. Parámetros del árbol de sumadores basado en olDFAs para 42 operandos 28

3.2. Parámetros del árbol de sumadores basado en olDFA$_{\mathrm{p}}$s para 42 operandos 29

3.3. Retardo online de árboles de sumadores olDFAs y olDFA$_{\mathrm{p}}$s 31

3.4. Códigos RBCD para la suma decimal multiformato 35

3.5. Retardo de la conversión a RBCD de los códigos presentes en la tabla 36

3.6. Comparación de los códigos RBCD y RBCD$_{-5421}$ 42

3.7. Resultados de olDFA$_{Mformat}$ no segmentado en el primer escenario de simulación (S1) 50

3.8. Resultados de olDFA$_{Mformat}$ segmentado en el primer escenario de simulación (S1) 50

3.9. Resultados de olDFA$_{Mformat}$ no segmentado en el segundo escenario de simulación (S2) 51

3.10. Resultados de olDFA$_{Mformat}$ segmentado en el segundo escenario de simulación (S2) 52

UNIVERSIDAD DE MÁLAGA

3.11. Resultados de olDFA en ambos escenarios de simulación (S1 y S2) 52

3.12. Resultados de olDFA$_p$ en ambos escenarios de simulación (S1 y S2) 53

4.1. Tabla de verdad del módulo multiplicador para la multiplicación del dígito x_k con los dígitos 2, 3 y 4 74

4.2. Tabla de verdad del módulo multiplicador para la multiplicación del dígito x_k con los dígitos 5, 6 y 7 75

4.3. Resultados del multiplicador decimal online sin especulación y con especulación 77

4.4. Resultados del multiplicador online y el multiplicador paralelo . . . 77

5.1. Constantes de selección 86

5.2. Resultados de la división online 88

A.1. Combinaciones válidas para el grupo v 96

A.2. Resto de combinaciones válidas para el grupo v 97

A.3. Combinaciones válidas para el grupo z 97

A.4. Numeración de las descomposiciones de u_i 97

A.5. Resumen del análisis de v_i con todas las codificaciones 98

A.6. Codificaciones válidas para cada descomposición de u_i 99

A.7. Resto de codificaciones válidas para cada descomposición de u_i . . 99

A.8. Camino crítico para cada modelo sin el módulo de descomposición 100

A.9. Codificaciones válidas para cada modelo 101

A.10. Resto de codificaciones válidas para cada modelo 102

A.11. Resultados Experimentales 104

C.1. Tabla de verdad para v 110

C.2. Tabla de verdad para v_{neg} 111

UNIVERSIDAD DE MÁLAGA

1 Introducción

Los ordenadores representan, almacenan y manipulan datos numéricos en formato binario. Muchas aplicaciones comerciales como el análisis financiero, banca, cálculo de tasas y operaciones contables se realizan utilizando aritmética binaria, que introduce un error de precisión al convertir un número decimal a binario y viceversa. Por ejemplo, considerese un cálculo donde se incluye un 5 % de tasa en un producto (como una llamada telefónica de 0.70€) y después se redondea el resultado al céntimo más cercano. Usando punto flotante binario con doble precisión, el resultado de multiplicar 0.70 por 1.05 es 0.73499999999999999€; sin embargo, el cálculo exacto es 0.735€. Usando números decimales el número se redondearía a 0.74€ (redondeo a par en el punto medio), pero usando binario el resultado devuelto sería el valor 0.73€, que es claramente incorrecto [5][1]. Por lo tanto, la aritmética decimal es una alternativa para contrarrestar la pérdida de precisión.

La mayoría de los procesadores de propósito general no proporcionan instrucciones o soporte hardware para aritmética de punto flotante decimal (DFP). Como resultado de ello, existen dos alternativas para mitigar la posible pérdida de precisión: a) los números decimales son leídos, convertidos a binario y se procesan utilizando la aritmética de punto flotante binario, aceptando la consiguiente pérdida de precisión; b) El uso de diferentes librerías software que trabajan sin pérdida de precisión. El primer enfoque proporciona los mejores resultados en velocidad, y es la opción que se utiliza para la mayoría de aplicaciones, mientras que el segundo se utiliza en los casos más críticos, pero con penalización de tiempo con respecto a la estrategia anterior. Con la demanda y crecimiento de aplicaciones de aritmética decimal en los recientes años, se han propuesto varias técnicas para resolver el problema de precisión, existendo soluciones que vienen

UNIVERSIDAD DE MÁLAGA

por dos vertientes: hardware y software. En términos de software, existen lenguajes de programación, como COBOL, XML, Visual Basic, Java, C que dan soporte a la aritmética decimal. Las soluciones hardware aparecieron en las décadas 50 y 60 al comienzo de los ordenadores digitales como ENIAC y UNIVAC [14]. Sin embargo, recientemente hay arquitecturas como IBM Power6, Power7, IBM z9, IBM z10, Fujitsu, SparcX [7, 19, 6, 30, 43], IP cores decimales [32] y diseños hardware [16, 36] que incluyen aritmética decimal en punto flotante.

1.1. Formato Decimal en Punto Flotante

En 2008 se terminó de introducir especificaciones y operaciones para números decimales en punto flotante (DFP) en el estándar IEEE 754. A partir de entonces dicho estándar se conoce como IEEE 754-2008. En él se definen dos formatos básicos de DFP, decimal64 y decimal128, con una longitud de codificación de 64 y 128 bits respectivamente. Los números decimales se codifican mediante un coeficiente y un exponente. Los coeficientes pueden ser representados usando codificación binaria o decimal. Con la codificación decimal el coeficiente se almacena usando la representación Densely Packed Decimal (DPD)[4], la cual puede ser convertida a Binary Coded Decimal (BCD) para realizar operaciones aritméticas.

Dentro cada formato se pueden representar los siguientes datos en punto flotante:

- Dos infinitos $+\infty$ y $-\infty$.
- Dos NaNs: qNaN y SNaN (no son números, como por ejemplo, la raíz cuadrada de un número negativo).
- Números DFP finitos.

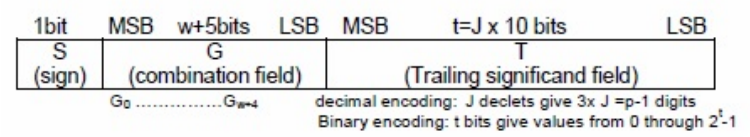


Figura 1.1: Formato Decimal en Punto Flotante (DFP).

Los datos decimales en punto flotante son codificados en k bits en tres campos como se muestra en la figura 1.1.

UNIVERSIDAD DE MÁLAGA

Parámetro	Decimal64	Decimal128
bias (valor)	398	6176
Signo (nº bits)	1	1
w+5, Longitud campo de combinación en bits	13	17
T, Longitud campo de mantisa en bits	50	110
K, almacenamiento en bits	64	128

Tabla 1.1: Parámetros de los formatos decimales decimal64 y decimal128

El primer campo es de 1 bit y codifica el signo S, donde el 0 representa un valor positivo y el 1 representa un valor negativo.

El campo de combinación G con w+5 bits (véase la tabla 1.1) codifica el exponente para los números finitos y los casos especiales.

El campo de la mantisa es de J x 10 bits y contiene parte del coeficiente (C). Cuando este campo se combina con los bits del campo de combinación G, el formato codifica un total de $p = 3 \times J + 1$ dígitos decimales y se obtiene el coeficiente.

Los valores de los números finitos decimales se calculan como: $(-1)^S \times 10^{E-bias} \times C$.

Los valores de los parámetros k, t, w y bias para los distintos formatos decimales aparecen en la tabla 1.1.

1.2. Aritmética Online

La aritmética online opera en serie, empezando desde el dígito más significativo (MSD) [8]. Además, la aritmética online tiene propiedades interesantes, como permitir el solapamiento de operaciones dependientes, lo que es una ventaja incluso para la multiplicación y comparación ya que los MSDs son producidos primero [8]. Además, la propuesta en serie reduce el área de los diseños y, generalmente, el consumo de energía, comparado con un operador paralelo. La desventaja de la aritmética online es el número de ciclos requeridos para realizar una operación. Sin embargo, esto puede ser compensado mediante el solapamiento de la ejecución de operaciones dependientes.

La aritmética online usando representación binaria tiene una amplia variedad de aplicaciones como son: filtros digitales [33], procesamiento de señales [9, 25], sistemas de comunicación wireless [29] y redes neuronales [13]. Por ello, con estas propiedades y aplicaciones, la aritmética online genera una posible solución de

UNIVERSIDAD DE MÁLAGA

Dígito	RBCD	Dígit	RBCD
0	0000		
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001

Tabla 1.2: Dígitos RBCD

diferentes algoritmos, lo cual la hace muy interesante. El interés por la aritmética decimal y la aritmética online durante los últimos años, abre un camino de investigación mezclando ambas aritméticas, resultando en aritmética decimal online.

En la aritmética online la representación más frecuentemente usada es de tipo Signed-Digit (SD), con un rango de dígitos simétrico $\{-a, ..., a\}$ y asimétrico $\{b, ..., c\}$. Las tres representaciones más usadas en Signed-Digit son: El código Svoboda [34], la representación componente positivo/negativo [16, 24, 26] y la representación Complemento a dos (C2). La forma de operar de izquierda a derecha de la computación online requiere una flexibilidad que se consigue mediante el uso de una representación redundante. Para el caso decimal, consideremos el caso general de un código decimal de 4-bits. Sea $[\alpha, \beta]$ el conjunto de valores de este código decimal. La condición de este código para tener suficiente redundancia para prevenir una propagación del acarreo es que $11 \leq \alpha + \beta \leq 15$ [16].

RBCD es un sistema numérico redundante decimal balanceado con un rango de dígitos simétrico $\{-7, \cdots, 7\}$ y que verifica la condición anterior. Un dígito RBCD se almacena con 4 bits y se representa en Complemento a dos. La tabla 1.2 muestra el rango y la codificación RBCD.

1.3. Motivación de la Tesis y Contribuciones

La actualización del estándar IEEE 754-2008 hizo que la aritmética decimal esté ahora más presente que nunca, dando lugar a multitud de diseños de unidades aritméticas decimales específicas. Alguno de los diseños más destacados son los de los procesadores IBM Power6, Power7, z9 y z10 que incluyen unidades decimales en punto flotante.

Si pasamos de la representación numérica al cálculo computacional, el estudio

UNIVERSIDAD DE MÁLAGA

de la precisión adquiere un nivel aún más complejo, ya que sucesivas operaciones aritméticas pueden producir errores de aproximación muy graves aun habiendo partido de representaciones numéricas con una exactitud bastante rigurosa. Es el caso de las operaciones financieras. En [1] se publicó un estudio acerca de los graves errores en que se puede llegar a incurrir en las transacciones financieras si la aritmética usada es la binaria. Además, aportan las modificaciones adecuadas a las arquitecturas contemporáneas para conseguir el soporte apropiado para aplicaciones financieras, todo ello cumpliendo con el estándar IEEE 754. Básicamente consiste por un lado, en mantener la conversión a binario de la parte entera para realizar las operaciones y transacciones, y por otro, en reajustar la parte fraccionaria tomando la representación binaria que se aproxime lo máximo posible y además genere el mínimo error teniendo en cuenta la sucesión de cálculos a realizar.

Aun resolviendo en gran medida la precisión que se necesita, la conversión de números decimales a formatos más manejables sigue siendo un paso intermedio en el proceso. Es por ello que surge la necesidad de encontrar un formato con el que sea especialmente viable la conversión decimal. En los últimos años, y con la aprobación del estándar IEEE 754-2008, muchas investigaciones han seguido esa línea.

En esta tesis se estudia la unión de la aritmética decimal y la artimética online para obtener un sistema online para operar con dígitos decimales usando la codificación RBCD que cumple con los requisitos de ambas aritméticas.

Las principales contribuciones de esta tesis son las siguientes:

- un *sumador decimal online (olDFA)* que realiza la suma de dos números RBCD utilizando un método de descomposición de mínima latencia. A su vez se presenta una versión del olDFA segmentada de 3 etapas (olDFA$_p$) para reducir el tiempo de cálculo. Mediante un estudio del procesado del stream de datos, se propone una solución para obtener el máximo throughput teórico posible en un sumador online. Finalmente, se realiza una comparativa estadística de los resultados de simulación de los dos diseños propuestos con sumadores paralelos que utilizan la codificación RBCD.

- un *sumador decimal online multioperando* definiendo un método para construir árboles con olDFAs y olDFA$_p$s como elementos base. También presentamos expresiones analíticas de las arquitecturas que resultan útiles en los estudios previos de los sistemas a diseñar. Los diseños presentados son comparados siguiendo criterios específicos de simulación y estudiando los resultados obtenidos en retardo y área.

UNIVERSIDAD DE MÁLAGA

- dos *estrategias para diseñar sumadores decimales online multioperando y multiformato*. La primera estrategia se basa en utilizar un olDFA con una etapa de conversión, y la segunda se basa en el diseño específico del sumador multiformato modificando, para ello, la arquitectura interna del olDFA. Ambas estrategias son comparadas en área y retardo siguiendo los mismos criterios de simulación.

- un *multiplicador decimal online* que sigue el algoritmo de multiplicación decimal online usando la codificación RBCD, que se basa en el uso de un residuo acumulativo recurrente, obteniendo el dígito del producto empezando por el bit más significativo (MSD). El residuo generado cada ciclo será utilizado en los futuros ciclos para compensar el error producido debido a la falta de datos característica de la aritmética online. Se presentan dos arquitecturas para comparar dos formas distintas de implementación del algoritmo, una de ellas utilizando especulación. A su vez se ha diseñado un multiplicador decimal online RTL de 16x16 dígitos, y se ha insertado en un sistema online. Por motivos de comparación, también hemos implementado el mismo sistema pero sustituyendo nuestro multiplicador online por un multiplicador decimal paralelo rápido.

- un *divisor decimal online*, que sigue un algoritmo basado en el uso de un residuo que se utiliza para compensar el error producido por la ausencia de todos los datos de entrada en cada ciclo (característico de la aritmética online). Para ello, se implementa un módulo de corrección que realiza la multiplicación vector por dígito de los dígitos obtenidos en los ciclos anteriores con el residuo acumulado. Dicho algoritmo se basa en la estrategia de separar el valor del cociente q en dos variables q_H y q_L. Se implementa a su vez una función de selección para obtener cada una de las variables del cociente que se basa en el uso de constantes de selección.

Las contribuciones anteriormente listadas han sido publicadas en conferencias internacionales [10, 12] y en revistas [11] clasificadas por el ISI Journal Citation Reports (JCR).

1.4. Estructura de la Tesis

El resto de la tesis está estructurada de la siguiente manera:

- El *capítulo 2* presenta los fundamentos del sumador decimal online (olDFA) utilizando la codificación RBCD y basándose en la descomposición de

UNIVERSIDAD DE MÁLAGA

las entradas. Se presenta una versión del olDFA segmentada de 3 etapas (olDFA$_p$) para reducir el tiempo de cálculo y se realiza una optimización en el procesamiento del stream de datos.

- En el*capítulo 3* se define un método para construir árboles de suma decimal online multioperando y se presentan expresiones analíticas de las arquitecturas que resultan útiles para realizar estudios previos de los sistemas a diseñar. También se propone dos diseños para realizar sumas decimales online multiformato, una de ellas con una etapa de conversión, y la segunda modificando la arquitectura interna del olDFA. Ambos diseños se presentan también con un estudio de sus versiones segmentadas.

- El *capítulo 4* presenta un algoritmo para la multiplicación decimal online basado en recurrencia del residuo y se expone el diseño de dos arquitecturas. La primera de ellas consiste en una arquitectura de multiplicación decimal online sin especulación y la segunda en una multiplicación decimal online con especulación con el objetivo de reducir el alto coste de computación en cada ciclo. Ambas arquitecturas son comparadas siguiendo unos parámetros de simulación.

- El *capítulo 5* presenta el algoritmo para realizar una división decimal online utilizando la códificación RBCD. Dicho algoritmo se basa en la estrategia de separar el valor del cociente q en dos variables q_H y q_L. Debido a que no se dispone de todos los datos, el algoritmo va generando un error que es compensado mediante el módulo de corrección. Por útlimo, se exponen los resultados en retardo y área obtenidos mediante la simulación de la implementación del algoritmo diseñado.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

2 Suma Decimal Online

En este capítulo se presenta el sumador decimal online (olDFA) que hemos realizado. Este sumador realiza la operación sobre dos operandos codificados en RBCD. El capítulo comienza con un resumen de los trabajos en los que nos hemos basados para diseñar el olDFA. Nuestra propuesta se basa en la serialización del sumador decimal paralelo presentado en [16] ya que era el sumador decimal más rápido de la literatura hasta la fecha. Éste, a su vez, se basa en el sumador decimal presentado en [31].

2.1. Sumador Paralelo RBCD

En [31] los autores proponen el diseño de un sumador paralelo RBCD basado en el algoritmo presentado en [2]. Para ello, los autores definen un mapeado de la suma de dos dígitos RBCD, x e y, como $F(x, y) = (C, S)$. La expresión de la función suma es $x + y = 10 * C + S$, donde C es el acarreo (transfer) para el siguiente dígito y S es el dígito del resultado de la suma. C está dentro del rango $\{-1, 0, 1\}$ y S está dentro del rango $\{-6, -5, \cdots, 5, 6\}$. Por ejemplo, la suma de los dígitos RBCD, $x = -7$ y $y = -1$, sería $x + y = \overline{1}2$ (el dígito $\overline{d}$ representa al dígito -d en RBCD).

El algoritmo para realizar la suma de dos dígitos RBCD sigue los siguientes pasos:

Paso i. Calcular la suma binaria $s = x + y$.

Paso ii. Detectar si hace falta realizar corrección en la suma binaria s y extraer el transfer para el siguiente dígito.

UNIVERSIDAD DE MÁLAGA

Paso iii. Calcular el dígito final de la suma. Al mismo tiempo, realizar la corrección y suma del transfer del dígito previo.

En el paso i, se calcula la suma binaria de x e y usando un sumador binario de 4-bits. Como RBCD es un sistema numérico redundante decimal con dígitos dentro del intervalo [-7,7], la suma binaria s esta dentro del rango [-14,14]. Es por ello que hay resultados que necesitan ser corregidos para ser representados como números RBCD. En el paso ii, se comprueba la suma binaria s para detectar si es necesaria una corrección y para ello se tiene en cuenta los siguientes casos:

a) si $s > 6$, se corrige la suma binaria sumando 6 y se genera un transfer $C = 1$ para la suma del siguiente dígito,

b) si $s < -6$, se lleva a cabo la corrección sumando -6 a s y se genera un transfer negativo $C = -1$ para la suma del siguiente dígito,

c) si s está dentro del rango [-6,6], no es necesaria corrección alguna y $C = 0$.

Las ecuaciones para detectar si s es mayor que 6 y la correspondiente generación del transfer $C = 1$ son las mismas ($f_6 = f_{C=1}$), y se muestra en la ecuación 2.1. Lo mismo ocurre con las ecuaciones para detectar si s es más pequeño que -6 y la generación de un transfer negativo $C = -1$ ($f_{\overline{6}} = f_{C=\overline{1}}$), y cuya expresión se muestra en la ecuación 2.2.

$$f_6 = f_{C=1} = \overline{x_3 y_3}(s_3 + s_2 s_1 s_0) \tag{2.1}$$

$$f_{\overline{6}} = f_{C=\overline{1}} = x_3 y_3(\overline{s_3} + \overline{s_2 s_1 s_0}) + (x_3 + y_3)(s_3 \overline{s_2 s_1 s_0}) \tag{2.2}$$

La razón para detectar $s > 6$ o $s < -6$ es dejar espacio para el transfer entrante de la suma del dígito anterior, evitando un transfer y paso de corrección extra en los sucesivos dígitos. En el paso iii del algoritmo, la suma binaria s es corregida al mismo tiempo que se añade el transfer producido en la suma del dígito anterior. La corrección se realiza añadiendo 6, -6 o 0 dependiendo del caso. Como el transfer del dígito de suma puede ser $\{-1, 0, 1\}$, hay 9 números posibles a añadir a la suma binaria s que son: $\{-7, -6, -5, -1, 0, 1, 5, 6, 7\}$. La suma de correción se lleva a cabo usando otro sumador binario de 4-bits.

Una vez realizado el paso de corrección, se obtiene el dígito correcto de la suma s.

UNIVERSIDAD DE MÁLAGA

La figura 2.1 muestra el hardware correspondiente al sumador RBCD paralelo [31]. Hay dos sumadores binarios de 4-bits: uno para realizar la suma binaria y otro para realizar la corrección y suma del transfer del dígito anterior. El bloque lógico llamado 'Detection' calcula las condiciones de corrección (f_6 y $f_{\overline{6}}$) y además, genera el transfer para el siguiente dígito ($f_{C=1}$ y $f_{C=\overline{1}}$). El bloque lógico 'Correction' genera la cantidad (w) necesaria para obtener la suma correcta, la cual se obtiene con el segundo sumador binario.

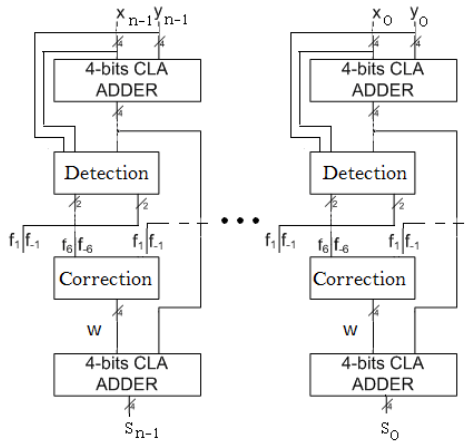


Figura 2.1: Sumador RBCD Paralelo.

2.2. Sumador DSSD

En [16] los autores describen una mejora del sumador RBCD y lo llaman sumador Digit Septa Signed Digit (DSSD). Su enfoque se basa en [31], pero la suma de los dos dígitos RBCD se interpreta como un número carry-save en complemento a dos. Los autores de [16] realizan una descomposición de los operandos para obtener una mejora en la suma. Esta descomposición se basa en tratar los pesos de los bits con valores positivos y negativos, lo que denominan como posibits y negabits [18]. Por ejemplo, considérese un sistema numérico, donde los bits en las posiciones 0 y 2 son posibits (y por lo tanto tienen peso positivo), y el bit en la posición 1 es un negabit (tiene peso negativo). Entonces, dada la combinación (111), el valor del número que representa dicha combinación sería $2^2 - 2^1 + 1 = 3$.

La suma de diferentes combinaciones de entrada de posibits y negabits tienen diferente resultados. Las Tablas 2.1 y 2.2 muestran las tablas de verdad de todas

UNIVERSIDAD DE MÁLAGA

las diferentes combinaciones de posibits y negabits.

X^+	Y^+	Z^+	C^+	S^+
X^-	Y^-	Z^-	C^-	S^-
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabla 2.1: Tabla de verdad para la suma 3 posibits y 3 negabits

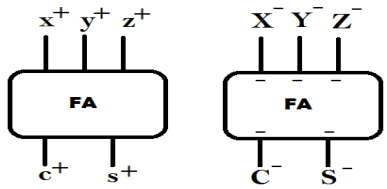
X^-	Y^-	Z^+	C^-	S^+
X^+	Y^+	Z^-	C^+	S^-
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Tabla 2.2: Tabla de verdad para sumar posibits y negabits mezclados

De las Tablas 2.1 y 2.2, podemos observar que las ecuaciones de un Full Adder (FA) con 3 entradas de posibits son las mismas que para un FA con 3 entradas de negabits. Las salidas de carry y suma son ambas posibits y negabits respectivamente. De la misma manera, las ecuaciones de un FA con entradas de dos negabits y un posibit son la misma que un FA con entradas de dos posibits y un negabit. Sin embargo, la interpretación de las salidas es diferente. Las ecuaciones de dichos FAs y su representación se describen en la figura 2.2 y en la figura 2.3. Dichos FAs especializados y algunos HAs se usan en el diseño del sumador DSSD. Nosotros nos basamos en estos sumadores para el diseño de nuestro sumador decimal online olDFA.

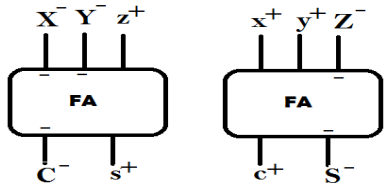
La descomposición de la suma redundante decimal propuesta en [16] se muestra en la figura 2.4. La primera y segunda fila representan los dos números RBCD a ser sumados, x y y. Como se puede ver, se representan como posibits (círculos negros con minúsculas) y negabits (círculos blancos con mayúsculas). El superíndice (subíndice) indica la posición del bit (dígito) con peso 2 (10). También

UNIVERSIDAD DE MÁLAGA



$$S = X \oplus Y \oplus Z$$
$$C = ((X \oplus Y) \cdot Z) + (X \cdot Y)$$

Figura 2.2: FAs con entradas de 3 posibits y 3 negabits.



$$S = X \oplus Y \oplus Z$$
$$C = ((X + Y) \cdot \overline{Z}) + (X \cdot Y)$$

Figura 2.3: Posibles casos de FA con posibits y negabits mezclados.

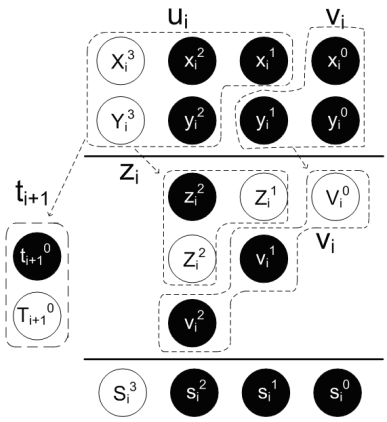


Figura 2.4: Descomposición de u_i y v_i.

se muestra en la figura 2.4 el proceso de descomposición. Este proceso consiste en agrupar los bits de los operandos x e y tal y como se muestra en la figura, con lo que se obtienen dos grupos de bits: u_i y v_i. La suma del grupo de bits u_i

UNIVERSIDAD DE MÁLAGA

puede tomar valores dentro del rango $\{-16, -14, \cdots, 8, 10\}$ y la del grupo de bits en v_i puede tomar valores dentro del intervalo $[0, 4]$. Además, estos dos grupos se descomponen/agrupan de nuevo. Del grupo u_i se extrae el grupo t_{i+1} y z_i. El grupo t_{i+1} es el transfer al siguiente dígito y está compuesto de un posibit (t_{i+1}) y un negabit (T_{i+1}) con el mismo peso. Los autores llaman a este grupo como el dígito de transferencia.

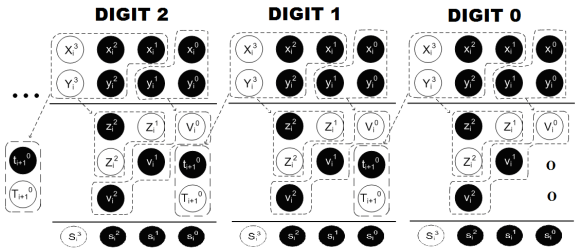


Figura 2.5: Operación paralela DSSD.

El grupo v_i se recodifica de otra manera para simplificar el siguiente paso de la descomposición de la suma. En este paso, se añade el dígito de transferencia de la suma del dígito anterior. En la figura 2.5 se puede observar como el sumador DSSD opera y cómo los dígitos de transferencia son enviados al siguiente dígito. Las ecuaciones para obtener los grupos de bits t_i, u_i y v_i son las siguientes:

$$
\begin{aligned}
t_{i+1} &= \overline{X_i^3 + Y_i^3} \\
T_{i+1} &= X_i^3 \cdot Y_i^3 \cdot (\overline{x_i^2} + \overline{x_i^1 \cdot y_i^2}) + (\overline{x_i^2 + y_i^2}) \cdot (\overline{x_i^1 \cdot (X_i^3 + Y_i^3)}) \\
Z_i^2 &= X_i^3 \cdot Y_i^3 \cdot (x_i^2 \cdot x_i^1 + \overline{y_i^2}) + \overline{X_i^3 \cdot Y_i^3} \cdot \overline{x_i^1} \cdot y_i^2 + \\
&\quad \overline{(X_i^3 + Y_i^3) \oplus x_i^1 \cdot \overline{y_i^2}} + (\overline{X_i^3} + \overline{Y_i^3}) \cdot x_i^2 \cdot \overline{y_i^2} \\
z_i^2 &= \overline{X_i^3 \cdot Y_i^3} \cdot (\overline{x_i^2 + y_i^2} \cdot \overline{x_i^1} + x_i^2 \cdot y_i^2) + X_i^3 \cdot x_i^2 \cdot (x_i^1 \oplus Y_i^3) + \\
&\quad \overline{y_i^2} \cdot (\overline{X_i^3 + Y_i^3} \cdot \overline{x_i^2} + x_i^2 \cdot x_i^1 \cdot Y_i^3) \\
Z^1 &= (\overline{x_i^2 + y_i^2} + x_i^1) \cdot (X_i^3 \oplus Y_i^3) + \\
&\quad \overline{X_i^3} \cdot (\overline{x_i^2 + y_i^2} \cdot x_i^1 + \overline{x_i^1} \cdot \overline{Y_i^3} \cdot (y_i^2 + x_i^2)) + \\
&\quad Y_i^3 \cdot (x_i^2 \cdot x_i^1 \cdot y_i^2 + X_i^3 \cdot \overline{x_i^1}) \\
V_i^0 &= x_i^0 \oplus y_i^0 \\
v_i^1 &= y_i^1 \oplus (x_i^0 + y_i^0) \\
v_i^2 &= y_i^1 \cdot (x_i^0 + y_i^0)
\end{aligned}
\tag{2.3}
$$

UNIVERSIDAD DE MÁLAGA

Las ecuaciones 2.3 son ligeramente diferentes de las ecuaciones presentadas en [16]. Esto se debe a que encontramos algunos errores en [16], los cuales corregimos y verificamos conjuntamente con los autores.

2.3. Sumador Decimal Online

En esta sección se presenta el diseño de un sumador online que opera dos números RBCD. Este elemento básico será usado en un sistema online. El sumador online se obtiene mediante la serialización del sumador RBCD paralelo presentado en [16]. También se presenta la segmentación que hemos realizado para reducir el tiempo de ciclo del sumador. Además, con el objetivo de reducir a su valor mínimo teórico posible el throughput en un stream de datos, hemos introducido una mínima modificación del hardware y que se detallará más adelante. Y para finalizar la sección, presentamos los resultados de una implementación de un sumador online para 16 dígitos.

2.3.1. Sumador Decimal Online (olDFA) para RBCD

En [41] los autores proponen un sumador online, llamado olFA, que opera con números codificados en binario. Siguiendo la misma filosofía, hemos diseñado el sumador online que opera números decimales codificados en RBCD. Llamaremos *online Decimal Full Adder* (olDFA) a este elemento básico ya que tiene una funcionalidad similar al olFA binario.

En la figura 2.6.a presentamos un diagrama de bloque del sumador RBCD paralelo propuesto en [16]. La operación de la figura 2.6.a puede entenderse mejor si emparejamos esta figura con la figura 2.5. Por otro lado, la figura 2.6.b muestra la versión online que proponemos en este apartado para dos números en RBCD de n-dígitos. Como se ve en la figura 2.6, obtenemos el olDFA mediante la serialización de la arquitectura paralela.

Vamos a describir en detalle la estructura interna del olDFA, que se muestra en más detalle en la figura 2.7. El módulo 'Decomposition' es un circuito lógico que se encarga de realizar las ecuaciones 2.3 (realiza el proceso de la descomposición de la suma). Los diferentes registros $rZ2, rz2, rv2, rZ1, rv1, rv0$ de la figura introducen un ciclo de retardo para emparejar en el tiempo el transfer del siguiente dígito t_{i+1}, T_{i+1} (todas las entradas a HA1, FA1 y FA2 de la figura 2.7 tienen el índice $i+1$). Estos registros producen un retardo online de uno ($\delta = 1$). Los restantes

UNIVERSIDAD DE MÁLAGA

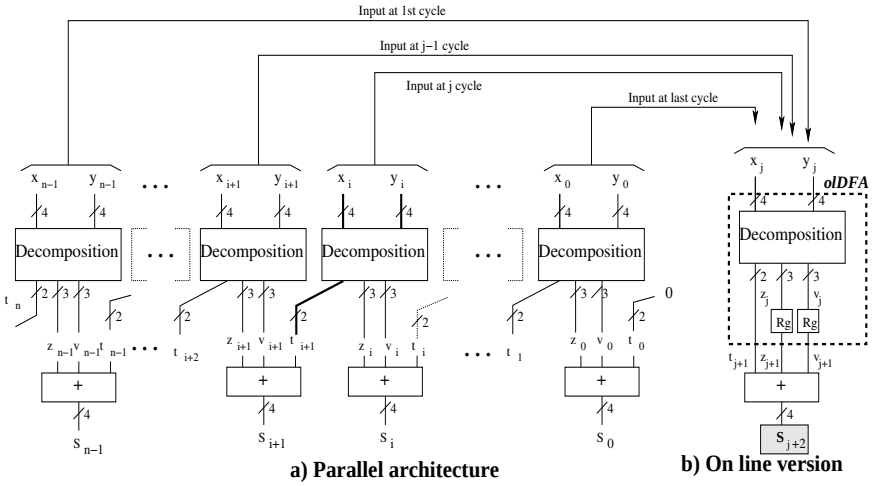


Figura 2.6: a) Estructura para sumador paralelo DSSD. b) Online decimal full adder (olDFA).

FA y HA son necesarios para obtener el dígito final S de acuerdo con el esquema de la figura 2.4.

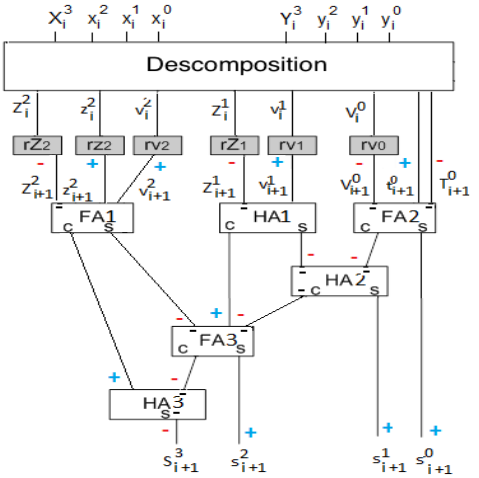


Figura 2.7: Estructura del olDFA.

Se pueden distribuir de otras formas los HAs y FAs de la figura obteniendo el mismo dígito final S y con el mismo retardo, pero se ha seleccionado la confi-

UNIVERSIDAD DE MÁLAGA

guración de la figura 2.7 porque permite una segmentación fácil del olDFA, tal y como se explicará en la sección 2.3.2.

En la figura 2.8 se muestra el diagrama de ciclo para realizar la suma de n-dígitos RBCD usando un olDFA. En el ciclo 0 se comienza el cálculo de los dígitos más significativos (MSDs) de los datos, (x_{n-1}, y_{n-1}), y en los siguientes ciclos se procesan los restantes dígitos. Después de la llegada de los últimos dígitos de los datos, (x_0, y_0), se necesita un ciclo extra debido al retardo online. Este ciclo extra permite a los dígitos menos significativos (LSDs) alcanzar el final del circuito (y así se vacía el camino de datos). En este ciclo *final* [23] se fuerzan los dos dígitos de entrada a 0, lo cual asegura que el resultado sea correcto. Debido a la estructura interna del olDFA esta operación no es trivial, pero se verá en detalle en la sección 2.3.3. En general, el número de ciclos *final* es igual al retardo online del operador [41] (i.e. si el retardo online del operador es de 3 por ejemplo, entonces se necesitan 3 ciclos *finales* para vaciar el camino de datos).

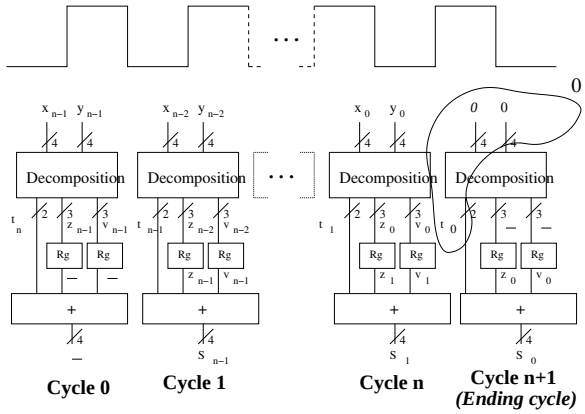


Figura 2.8: Suma de n-dígitos RBCD usando un olDFA.

2.3.2. Segmentación de la Arquitectura del olDFA

Vamos a denotar como $D_{decom}, D_{FA}, D_{HA}, D_{reg}$ al retardo del módulo de descomposición, de un FA, de un HA y de la carga de un registro, respectivamente. Si nos fijamos en la figura 2.7 vemos que el camino crítico del olDFA pasa a través del módulo de descomposición, FA2, HA2, FA3 y HA3. Por lo tanto, el ciclo de reloj del olDFA es:

UNIVERSIDAD DE MÁLAGA

$$CC_{\text{olDFA}} = D_{decom} + 2 \cdot D_{FA} + 2 \cdot D_{HA} + D_{reg} \tag{2.4}$$

Para reducir el ciclo de reloj, hemos realizado una segmentación de la arquitectura. La segmentación es una técnica usada para reducir el ciclo de reloj de un diseño dado, pero a costa de incrementar la latencia. Para ello se introducen registros intermedios a lo largo del diseño creando una serie de etapas por las que van pasando los datos a ser procesados. Por lo tanto insertamos dos niveles de registros de segmentación en la arquitectura de la figura 2.7 con lo que conseguimos obtener unas etapas bien balanceadas. La arquitectura del sumador online decimal segmentado (olDFA$_{\text{p}}$) se muestra en la figura 2.9. Se han seleccionado estas posiciones para los registros de segmentación ya que el retardo del módulo de descomposición es cercano al de un FA más un HA (el módulo de descomposición fue especialmente diseñado en [16] para minimizar el retardo). Con el diseño de la figura 2.9 el retardo de cada etapa es de 4 puertas lógicas aproximadamente (los valores de retardo de la implementación se presentan en la sección 2.3.4).

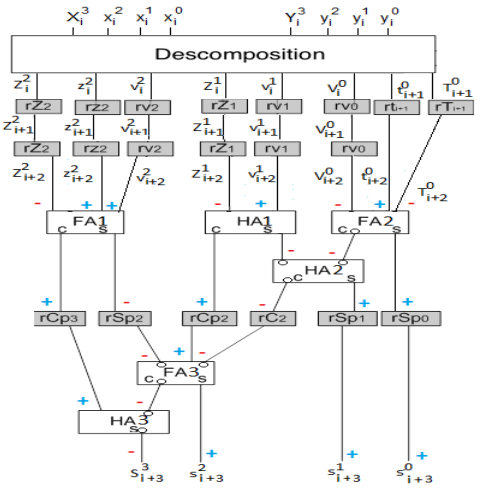


Figura 2.9: Arquitectura olDFA Segmentada (olDFA$_{\text{p}}$).

Con la arquitectura segmentada propuesta olDFA$_{\text{p}}$, el ciclo de reloj es:

$$CC_{\text{olDFA}_{\text{p}}} = max\{D_{decom}, D_{FA} + D_{HA}\} + D_{reg} \tag{2.5}$$

Dependiendo de la tecnología de la implementación (routing y latencia de las

UNIVERSIDAD DE MÁLAGA

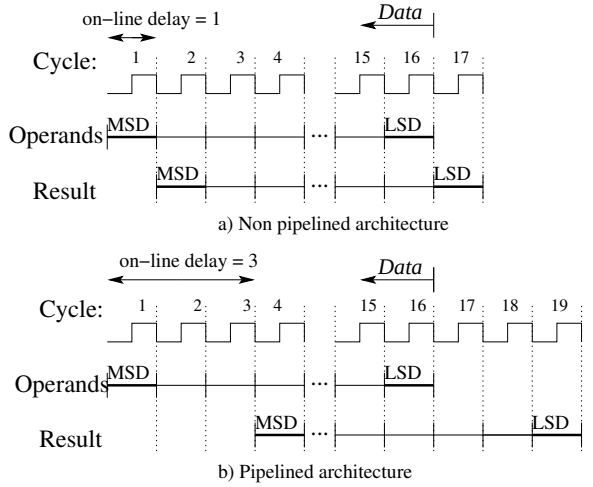


Figura 2.10: Sincronización para datos con $n = 16$ dígitos para olDFA y olDFA$_{\mathrm{p}}$.

diferentes puertas) el elemento dominante puede ser el módulo de descomposición o los elementos FA+HA.

El retardo online de la arquitectura segmentada es de tres ciclos, ya que es el tiempo para obtener el MSD del resultado. La figura 2.10 muestra la temporización para la arquitectura del sumador online segmentada y para la no segmentada.

2.3.3. Stream de Datos

Consideremos un stream de datos, donde cada dato está compuesto por números RBCD de n-dígitos. Este stream se aplica como entradas a un olDFA. El objetivo de esta subsección es estudiar el throughput del sistema y proporcionar una técnica para reducirlo. El throughput está directamente relacionado con el intervalo de iniciación entre dos datos sucesivos. Una situación similar puede verse en [41] para la suma online binaria. Sin embargo, el caso decimal es diferente y la solución la presentamos a continuación.

El intervalo de iniciación mínimo teórico es igual al número de dígitos de los operandos, es decir, n. Sin embargo, el intervalo de iniciación del olDFA es mayor debido al retardo online de éste ($\delta = 1$), teniendo $n + 1$ ciclos de intervalo de iniciación entre dos datos consecutivos, como se muestra en la sección 2.3.1. La figura 2.10.a muestra la situación en el caso de tener datos con $n = 16$ dígitos. Podemos ver que el intervalo de iniciación es de 17 ciclos.

UNIVERSIDAD DE MÁLAGA

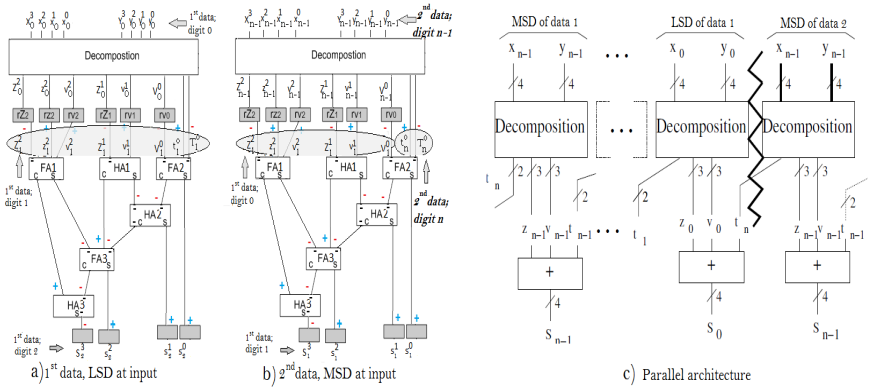


Figura 2.11: Transición entre dos datos para el olDFA y el sumador paralelo.

Debido al comportamiento de la aritmética online, las señales correspondientes a diferentes dígitos coinciden dentro del olDFA al mismo tiempo (dígito i y dígito $i+1$ en la figura 2.7). Esto no es un problema excepto cuando dos datos diferentes están involucrados. Para ser más específicos, es posible tener el LSD de un dato anterior y el MSD del dato siguiente dentro del olDFA en el mismo ciclo. Como consecuencia, las entradas del transfer del actual LSD son iguales al transfer de salida del MSD del dato siguiente.

Por ello, la salida del transfer del MSD (t_n, T_n) del nuevo dato debe ser independiente de la entrada del transfer del LSD del dato previo (t_0, T_0). La figura 2.11 muestra el estado de las diferentes señales para dos ciclos consecutivos sólo en la transición entre datos.

En la figura 2.11.a se puede observar la situación justo a la llegada de los LSDs del primer dato. Todas las entradas a HA1, FA1 y FA2 corresponden al dígito 1 (incluyendo los bits de transferencia (t_1, T_1)). En el siguiente ciclo (figura 2.11.b) los MSDs del segundo dato entran en el olDFA. Se observa que las entradas a HA1 y FA1 corresponden al dígito 0 del primer dato, pero dos de los bits de entrada a FA2 corresponden al bit de transferencia del MSD del segundo dato (t_n, T_n), el cual estaría produciendo un resultado erróneo (mezclando dos datos diferentes).

La arquitectura paralela equivalente mostrada en la figura 2.11.c puede ayudar a entender la situación, donde se puede observar que las señales (t_n, T_n) del segundo dato *cruzan* la barrera para ser las señales (t_0, T_0) del primer dato (los primeros transfer de entrada), que deben ser igual a cero para prevenir que influyan en la computación del primer dato.

UNIVERSIDAD DE MÁLAGA

En otras palabras, las salidas del transfer del MSDs del segundo dato están conectadas a las entradas del transfer del LSD del primer dato (por ejemplo (t_n, T_n) están conectados a (t_0, T_0)), lo cual es erróneo.

Para prevenir esta situación se tiene que asegurar que el primer y segundo dato sean independientes. La solución clásica a este problema es insertar un dígito extra 0 en ambas entradas del olDFA entre dos datos consecutivos. Por lo tanto, se necesita un ciclo de *separación* para aislar la computación de dos datos consecutivos. Esta acción fuerza a 0 los bits de transferencia. Debido a ello, el throughput corresponde a un intervalo de iniciación de $n + 1$ ciclos, como se muestra en la figura 2.12 para datos de 16 dígitos.

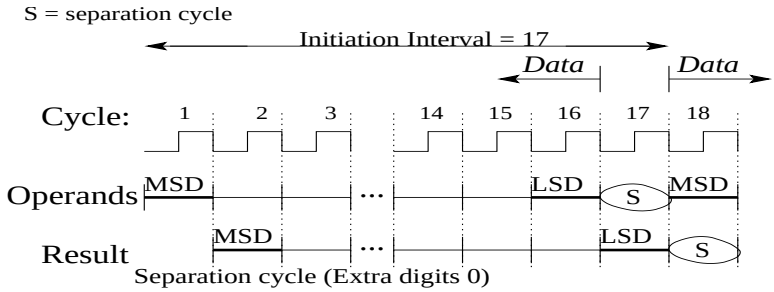


Figura 2.12: Intervalo de iniciación (se necesita un ciclo de separación para insertar un dígito 0 extra).

En el caso de la versión segmentada, el número de ciclos de separación se mantiene en uno y los ciclos de finalización son iguales a tres. Es decir, sólo se tiene que insertar un ciclo de dígitos con valor 0 entre dos datos consecutivos.

Mejora del throughput

En esta subsección proponemos una modificación hardware para reducir el intervalo de iniciación, obteniendo el mínimo teórico.

La idea clave es forzar el transfer del MSD del segundo dato a ser cero, es decir, hacer $t_n = 0, T_n = 0$ y por ello $t_0 = 0, T_0 = 0$. De esta manera, los bits del transfer usados como entradas del transfer al LSD del primer dato son cero y la computación es correcta.

Para hacer esto, insertamos una puerta AND en cada uno de los bits del transfer para controlar sus estados, como se muestra en la figura 2.13. La entrada de control de estas puertas es 1 para todos los ciclos de computación de un dato,

UNIVERSIDAD DE MÁLAGA

excepto para el primer ciclo, en el cual es forzada a 0. Por ello, cuando un dato nuevo entra al olDFA, las puertas son forzadas a cero durante el primer ciclo.

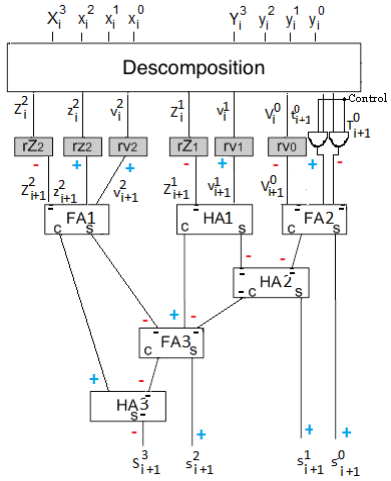


Figura 2.13: olDFA con control de los transfers de entrada.

Con el control directo de los bits del transfer propuesto en la figura 2.13 se reduce el intervalo de iniciación a su mínimo teórico: n ciclos para datos de n dígitos, como se muestra en la figura 2.14.

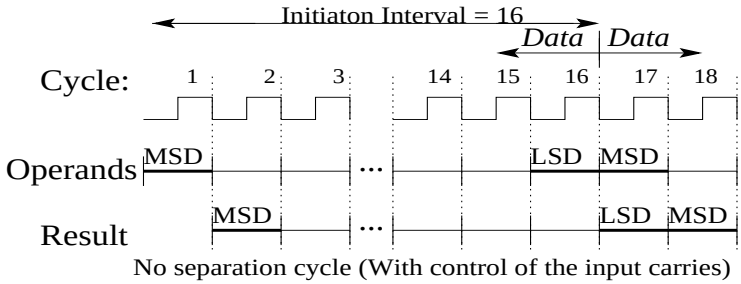


Figura 2.14: Intervalo de iniciación (sin ciclo de separación).

El aumento de velocidad es $\frac{n+\delta}{n}$ para un sistema que opera con números de n dígitos y un retardo online δ. En el olDFA para números de n=8 dígitos, el aumento de velocidad es del 12,5 % y si n=16 es del 6,25 %. El coste hardware para reducir el intervalo de iniciación es despreciable como se puede ver en el resultado de la implementación real (ver sección 2.3.4).

UNIVERSIDAD DE MÁLAGA

2.3.4. Resultados Experimentales y Comparación

Los diseños presentados en esta sección han sido implementados en Verilog, simulados usando ModelSim 6.0, y sintetizados usando Synopsys Design Compiler y la librería TSMC 65nm en la cual una unidad de celda tiene un área igual a 1 μm^2.

Para poder realizar una comparación hemos diseñado e implementado dos sumadores online basados en el sumador RBCD paralelo de [31]. La figura 2.15.a muestra la arquitectura del sumador online basado en el sumador propuesto en [31] sin segmentar y en la figura 2.15.b mostramos su versión segmentada.

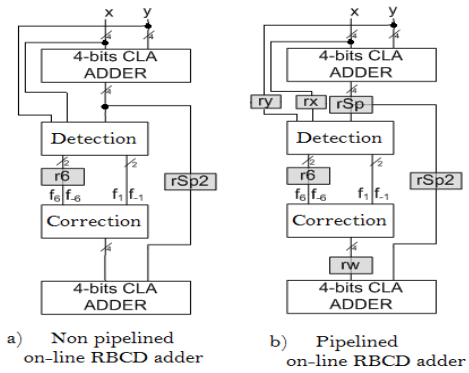


Figura 2.15: Sumadores online basados en el sumador RBCD paralelo de [31].

En la Tabla 2.3 presentamos los resultados de implementación para nuestros olDFAs y los de las arquitecturas mostradas en la figura 2.15. Hemos llamado *sumador RBCD online* a la arquitectura de la figura 2.15.a y *sumador RBCD online segmentado* a la de la figura 2.15.b. En la Tabla 2.3 se presentan el retardo, área y throughput de todas las arquitecturas implementadas. El throughput corresponde al caso correspondiente a operandos de $n = 16$ dígitos y proporciona el número de millones de sumas totales por segundo (cada suma total tiene 16 ciclos ya que el ciclo de separación entre datos no es necesario debido a la mejora propuesta).

Observando los resultados presentados en la Tabla 2.3, podemos ver que el olDFA es un 36 % más rápido que el *sumador RBCD online* (figura 2.15.a) y con un 20 % menos de área, mientras que para las versiones segmentadas, el olDFA$_p$ es un 28 % más rápido que el sumador RBCD online segmentado (figura 2.15.b) y con un 29 % menos de área.

UNIVERSIDAD DE MÁLAGA

Diseño	Retardo (ns)	Área (μm^2)	throughput n=16 (Millones de sumas enteras/s)
olDFA	0.22	395	284
olDFA_p	0.17	488	367
sumador RBCD online	0.3084	498	208
sumador RBCD online segmentado	0.2374	688	271

Tabla 2.3: Resultados experimentales de los sumadores decimales online

Si comparamos el olDFA con el olDFA_p vemos que el olDFA_p es un 23 % más rápido a pesar de tener tres etapas de segmentación. Esto es debido al hecho de que el retraso de la carga de los registros de segmentación es relativamente mayor comparado con el hardware de cada etapa (un resultado similar se puede observar para el sumador RBCD).

2.4. Conclusiones del Capítulo

En este capítulo se ha presentado el diseño de un elemento básico para construir sistemas decimales online. El olDFA propuesto suma dos números RBCD siguiendo la descomposición de mínima latencia propuesta en [16]. Para reducir el tiempo de cálculo, se ha presentado una versión segmentada del olDFA de 3 etapas (olDFA_p).

También se ha tratado el problema del procesado de stream de datos, proponiendo una solución para obtener el máximo throughput teórico posible en un sumador online. La inserción de dos puertas estratégicamente colocadas en el diseño, hace posible el intervalo de iniciación mínimo con un coste despreciable de hardware y de retardo.

Además se han analizado algunos ejemplos prácticos acordes a la precisión de números DFP *decimal64* (correspondiente a números de n=16 dígitos). Todos los diseños se han implementado y los resultados han sido presentados y discutidos en este capítulo. De los resultados obtenidos se ha visto que el olDFA propuesto es un 36 % más rápido que la versión online del sumador paralelo RBCD. En el próximo capítulo utilizamos el olDFA como un elemento básico para construir sumadores multioperando y multiformato.

UNIVERSIDAD DE MÁLAGA

3 Sumador Decimal Online Multioperando y Multiformato

En este capítulo presentamos un método para diseñar un sumador online decimal multioperando que utiliza el mínimo hardware posible.

Un sumador multioperando se diseña conectando sumadores de forma que se crea una estructura en forma de árbol. Por esta razón utilizaremos de forma indistinta los términos sumador decimal online multioperando y árbol de sumadores a lo largo del capítulo.

El sumador decimal multioperando opera sólo con números codificados en RBCD. Pero en la literatura existen otros diseños decimales que utilizan otros formatos decimales y que hacen posible la optimización de los algoritmos decimales implementados. Por ejemplo, en [40] el CORDIC decimal utiliza los formatos decimales con peso 5221, 5421 y en [39] el formato utilizado es 5211. Para los multiplicadores decimales de alto rendimiento presentados en [36] y en [38] los formatos utilizados son 4221 y 5421. Por lo tanto, el uso de diferentes formatos puede ser útil para diseñar arquitecturas específicas como las que han sido presentadas como ejemplos en [11] y que detallaremos en la sección 3.3.

Con el objetivo de poder realizar sumas con números que estén codificados con otros códigos distintos a RBCD, también presentamos en este capítulo un sumador online decimal multioperando y multiformato. Los resultados de las distintas implementaciones que se han propuesto se presentan y analizan a lo largo del capítulo.

UNIVERSIDAD DE MÁLAGA

3.1. Sumador Decimal Online Multioperando

Proponemos un método para construir árboles de sumadores basados en olDFAs y olDFA$_p$s que utiliza el mínimo hardware. Además, hemos realizado un estudio analítico con lo que podemos calcular de antemano diversos parámetros del sistema sin tener que implementar físicamente un árbol de sumadores.

3.1.1. Construcción de Árboles de Sumadores basados en olDFAs

Un sumador multioperando (ya sea paralelo u online) se construye a partir de sumadores básicos, conectando la salida de varios sumadores a la entrada de otro sumador [8]. De esta forma se van creando arquitecturas en forma de árbol. En nuestro caso, el sumador online básico es el olDFA presentado en el capítulo anterior. Uniendo olDFAs se pueden generar distintas arquitecturas que den como resultado un sumador decimal online multioperando para un número de operandos dado, m. Por ejemplo, la figura 3.1 muestra dos arquitecturas diferentes de árboles de sumadores basadas en olDFAs para $m = 6$ operandos.

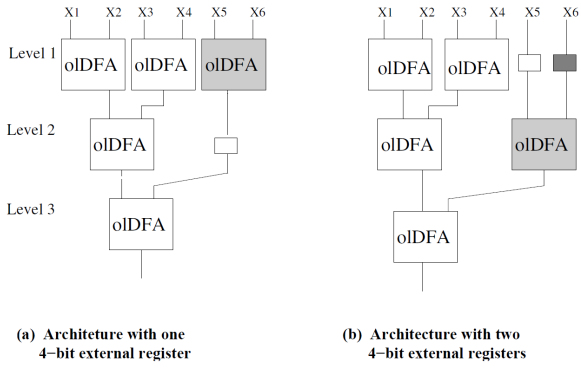


Figura 3.1: Dos arquitecturas basadas en olDFA para 6 operandos.

Comparando ambas arquitecturas, puede verse que la descrita en la figura 3.1.b realiza la suma de los operandos x_5 y x_6 en el segundo nivel del árbol. Y por ello, se necesita dos registros externos de 4 bits para asegurar la sincronización y obtener la suma correcta. Estos registros externos son necesarios porque cada módulo olDFA tiene un registro interno de 4 bits. La desventaja de la arquitectura mostrada en la figura 3.1.b es que tiene un registro externo más que la mostrada

UNIVERSIDAD DE MÁLAGA

en la figura 3.1.a, con un mismo tiempo de ejecución en ambas arquitecturas.

Por lo tanto, en términos de ahorro de hardware y consumo de energía, el mejor método para construir árboles de sumadores es generar arquitecturas donde en cada nivel se coloquen el máximo número de olDFAs posibles, empezando a completar siempre por el nivel superior. De esta forma se obtendrán los árboles con el mínimo hardware. Es de notar que el número de olDFAs para un mismo número de operandos es independiente de la configuración del árbol. Dicho de otro modo, todas las configuraciones de árboles posibles requieren el mismo número de olDFAs.

Si construimos los árboles de sumadores siguiendo este método y enumerando sus niveles tal y como se muestra en la figura 3.1.a, obtenemos las expresiones aritméticas de los parámetros que presentamos más abajo. Hay que tener en cuenta que todos los parámetros dependen del número de operandos, m.

Párametros en un árbol de sumadores online

El número total de olDFAs en el árbol de sumadores, k, se obtiene como $k = m - 1$.

El número de niveles del árbol de sumadores, L, es $L = \lceil log_2 m \rceil$.

El número de operandos en el nivel l, m_l, se define mediante la recurrencia $m_l = \lceil m_{l-1}/2 \rceil$ siendo $m_1 = m$.

De la última expresión, podemos calcular el número de módulos olDFAs y registros externos en el nivel l (k_l y R_l), cuyas expresiones son: $k_l = \lfloor m_l/2 \rfloor$, y $R_l = m_l \ mod \ 2$.

Estas expresiones puede ser útiles en el proceso de diseño de un árbol de sumadores basados en olDFAs, ya que pueden orientar sobre el consumo o área que pueda tener un sumador multioperando de m operandos. Por ejemplo, en la Tabla 3.1 se muestran los valores de los parámetros mencionados anteriormente de un árbol de sumadores de $m = 42$ operandos.

Otra expresión que se puede calcular de forma analítica es el ciclo de reloj de un árbol de sumadores basado en olDFAs de m operandos. El ciclo de reloj tiene la siguiente expresión:

$$\begin{aligned} CC_{olDFA_m} &= L \cdot (T_{decom} + 2 \cdot T_{FA} + 2 \cdot T_{HA} + T_{reg}) \\ &= L \cdot CC_{olDFA} = \lceil log_2 m \rceil \cdot CC_{olDFA} \end{aligned} \tag{3.1}$$

Con respecto al ciclo de reloj, se debe mencionar dos cosas: la primera es que depende del número de niveles del árbol (y por lo tanto, depende de m), y la

UNIVERSIDAD DE MÁLAGA

Nivel l	# Oper. m_l	# olDFA k_l	# 4-bit reg. ext. R_l
1	42	21	0
2	21	10	1
3	11	5	1
4	6	3	0
5	3	1	1
6	2	1	0

Tabla 3.1: Parámetros del árbol de sumadores basado en olDFAs para 42 operandos

segunda es que la expresión de CC_{olDFA_m} es válida para cualquier arquitectura basada en olDFAs (sin que se haya construido siguiendo el método anteriormente mencionado).

3.1.2. Construcción de Árboles de Sumadores basados en olDFA$_p$s

Ya que el ciclo de reloj de los árboles de sumadores basados en olDFAs crece con el número de operandos y CC_{olDFA_p} es menor que CC_{olDFA}, decidimos construir y analizar árboles de sumadores teniendo como componente el módulo olDFA$_\mathrm{p}$.

Los árboles de sumadores basados en olDFA$_\mathrm{p}$s se construyen también siguiendo el método presentado en la subsección anterior; es decir, colocando los máximos

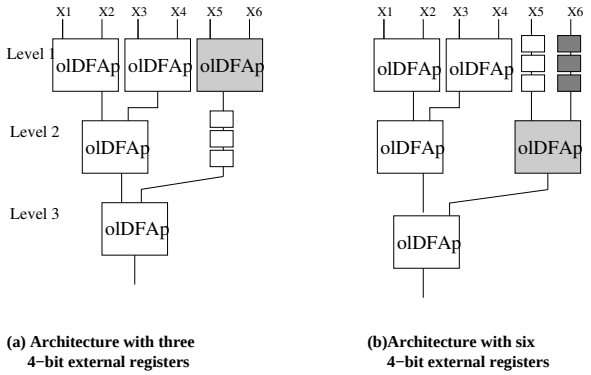


Figura 3.2: Dos arquitecturas basadas en olDFA$_\mathrm{p}$ para 6 operandos.

UNIVERSIDAD DE MÁLAGA

módulos olDFA$_p$s posibles en cada nivel empezando desde arriba. Esto se puede observar en el ejemplo descrito en la figura 3.2 donde la arquitectura de la figura 3.2.a tiene menos hardware que la mostrada en la figura 3.2.b.

Como en la subsección previa, se puede obtener algunas expresiones para el modelado de árboles de sumadores basados en olDFA$_p$s. De hecho, todas las expresiones son las mismas que las obtenidas para los árboles de sumadores basados en olDFAs, excepto la del parámetro R_l y el ciclo de reloj.

La expresión de R_l para un árbol de sumadores basado en olDFA$_p$s es:

$R_l = 3 \cdot (m_l \ mod \ 2)$

Como se puede observar, es tres veces la expresión de R_l para un árbol basado en olDFAs. Esto se debe a que $\delta_{olDFA_p} = 3$ (cada olDFA$_p$ tiene tres filas de registros internos para obtener la segmentación en tres etapas).

La Tabla 3.2 muestra algunos parámetros para un árbol de sumadores basado en olDFA$_p$s para $m = 42$ operandos. Como se puede ver, los valores de los parámetros son los mismos que en el caso del árbol basado en olDFAs para $m = 42$ operandos, excepto por el parámetro R_l el cual es 3 veces el valor del parámetro R_l del árbol basado en olDFAs.

Con respecto a la expresión del ciclo de reloj de un árbol de sumadores basado en olDFA$_p$s para m operandos, se tiene que:

$$CC_{olDFAp_m} = T_{decom} + T_{FA} + T_{HA} + T_{reg} \tag{3.2}$$

el cual es menor que CC_{olDFAm} y no depende del número de operandos. Como en el caso de los árboles de sumadores basados en olDFAs, la expresión CC_{olDFAp_m} es válida para cualquier arquitectura basada en olDFA$_p$s (sin que se haya construido siguiendo el método ya mencionado).

Nivel l	# Oper. m_l	# olDFA$_p$ k_l	# 4-bit reg. ext. R_l
1	42	21	0
2	21	10	3
3	11	5	3
4	6	3	0
5	3	1	3
6	2	1	0

Tabla 3.2: Parámetros del árbol de sumadores basado en olDFA$_p$s para 42 operandos

UNIVERSIDAD DE MÁLAGA

3.1.3. Árboles de Sumadores Segmentados

En esta subsección estudiamos el efecto de la segmentación de los árboles de sumadores presentados en las subsecciones anteriores.

Después de analizar diferentes opciones y como ocurría con el caso no segmentado, observamos que la colocación de los registros en cada nivel de un árbol de sumadores (tal y como se muestra en las figuras 3.3 y 3.4) es la mejor opción para obtener el menor retardo, y además, obtener unas etapas bien balanceadas.

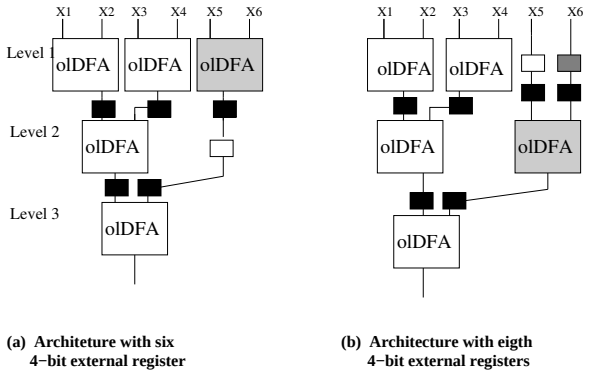


Figura 3.3: Dos arquitecturas segmentadas basadas en olDFAs para 6 operandos.

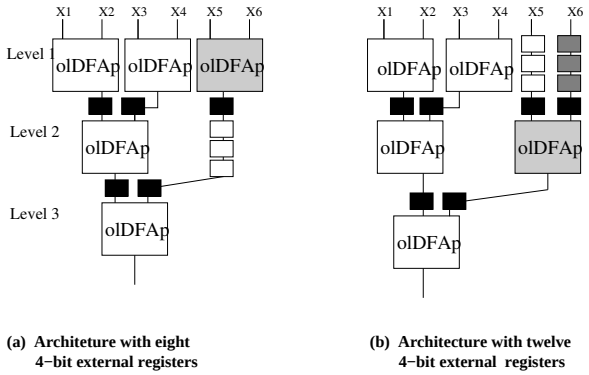


Figura 3.4: Dos arquitecturas segmentadas basadas en olDFA$_p$s para 6 operandos.

La figura 3.3 y la figura 3.4 muestran ejemplos de arquitecturas segmentadas basadas en olDFAs y olDFA$_p$s para $m = 6$ operandos. Las arquitecturas descritas en la figura 3.3.a y la figura 3.4.a son las mejores opciones en términos de menor

UNIVERSIDAD DE MÁLAGA

uso de hardware, como ocurría en el caso no segmentado.

Aunque el ciclo de reloj de las arquitecturas segmentadas disminuye con respecto al ciclo de reloj de la versión no segmentada, no pasa lo mismo con el retardo online, el cual se incrementa.

La Tabla 3.3 muestra el retardo online para todos los árboles de sumadores presentados hasta ahora (versiones segmentadas y no segmentadas). Puede verse que el retardo online de los árboles de sumadores basados en olDFA$_{\text{p}}$s (segmentados o no) es mayor que el de los árboles basados en olDFAs. Pero a pesar de ello, se muestra en la siguiente subsección que es mejor en términos de ciclo de reloj y throughput, construir árboles de sumadores multioperando usando olDFA$_{\text{p}}$s.

m	δ_{olDFA_m} L	δ_{olDFAp_m} $3 \cdot L$	$\delta_{P-olDFA_m}$ $2 \cdot L - 1$	$\delta_{P-olDFAp_m}$ $4 \cdot L - 1$
3	2	6	3	7
4	2	6	3	7
5	3	9	5	11
6	3	9	5	11
7	3	9	5	11
8	3	9	5	11
9	4	12	7	15
10	4	12	7	15
11	4	12	7	15
12	4	12	7	15

Tabla 3.3: Retardo online de árboles de sumadores olDFAs y olDFA$_{\text{p}}$s

En cuanto al ciclo de reloj de las arquitecturas segmentadas se tiene que la expresión para un árbol de sumadores segmentado basado en olDFAs para m operandos es:

$$\begin{aligned} CC_{P-olDFA_m} &= T_{decom} + 2 \cdot T_{FA} + 2 \cdot T_{HA} + T_{reg} \\ &= CC_{olDFA} \end{aligned} \tag{3.3}$$

y la expresión del ciclo de reloj de un árbol de sumadores segmentado basado en olDFA$_{\text{p}}$ para m operandos es:

$$\begin{aligned} CC_{P-olDFAp_m} &= max\{T_{decom}, T_{FA} + T_{HA}\} + T_{reg} \\ &= CC_{olDFAp}. \end{aligned} \tag{3.4}$$

UNIVERSIDAD DE MÁLAGA

Analizando ambas expresiones se puede ver que los árboles segmentados basados en olDFA$_p$s tienen el menor ciclo de reloj.

3.1.4. Resultados Experimentales

Todas las arquitecturas presentadas en esta sección se han modelado y verificado a nivel RTL con Verilog. También, se han sintetizado usando Synopsis Design Compiler y la libería de celdas TSMC's tcbn65gplus 65 nm CMOS. Todo el entorno y parámetros de proceso se fijaron a las condiciones normales o típicas.

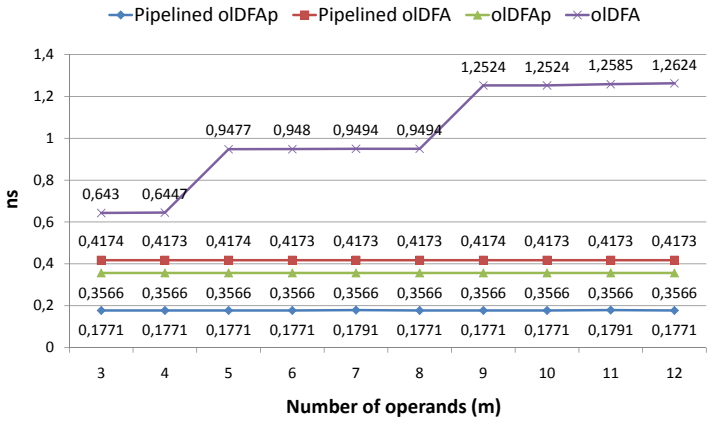


Figura 3.5: Ciclo de reloj (retardo) de las arquitecturas Multioperando basadas en olDFA y olDFAp segmentadas y sin segmentar.

En la figura. 3.5 mostramos el ciclo de reloj (retardo) de los sumadores multioperando propuestos en esta sección, para un rango de operandos. En el eje x se representa el número de operandos del árbol de sumadores y en el eje y se representa el ciclo de reloj en nanosegundos. Como se esperaba, el parámetro CC_{olDFA_m} va incrementándose con el número de operandos, m, mientras que los demás ciclos de reloj permanecen constantes (no dependen de m). Por otra parte, los resultados de sintetización verifican que CC_{olDFAp_m} es el menor ciclo de reloj de todas las arquitecturas propuestas.

En la figura 3.6 mostramos el área de las diferentes arquitecturas.

De nuevo, el eje x representa el número de operandos y el eje y representa el

UNIVERSIDAD DE MÁLAGA

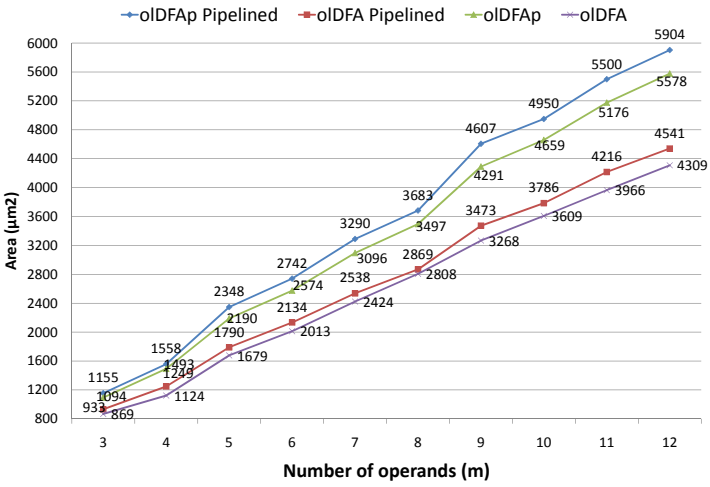


Figura 3.6: Área de las arquitecturas Multioperando basadas en olDFA y olDFAp segmentadas y sin segmentar.

área en μm^2. En este caso, el árbol de sumadores basados en olDFAs tiene menos área que el basado en olDFA$_p$s. Obviamente ésto se debe al hecho de que cada módulo olDFA$_p$ tiene dos filas de registros más que el árbol de sumadores basado en olDFAs para un número de operandos dado. Con estos resultados, se puede observar que el árbol de sumadores segmentado basado en olDFA$_p$s tiene entre un 24 % y un 30 % más de área que los basados en olDFAs.

Para analizar el efecto del control de hardware de los módulos olDFA y olDFA$_p$ en el throughput de las arquitecturas, la figura 3.7 presenta el throughput de todas las arquitecturas cuando operan con datos de $n = 16$ dígitos.

En la figura se puede observar que los árboles de sumadores segmentados basado en olDFA$_p$s tiene entre un 73 % y 86 % más de throughput que los basados en olDFAs. Con esta observación, podemos concluir que es mejor, en términos de ciclo de reloj y throughput, construir los árboles de sumadores usando los módulos olDFA$_p$s. Además, se ha probado que es más eficiente trabajar con árboles segmentados que tengan como componentes el módulo olDFA$_p$.

UNIVERSIDAD DE MÁLAGA

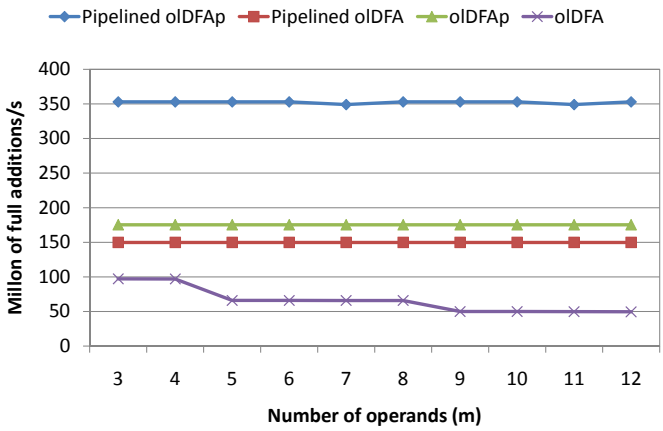


Figura 3.7: Throughput de las arquitecturas Multioperando basadas en olDFA y olDFAp y segmentadas y sin segmentar.

3.2. Sumador Decimal Online Multiformato

En esta sección describimos el diseño de sumadores decimales online que soportan entradas de operandos codificados con diferentes formatos (multiformato) y que proporcionan el resultado en el formato RBCD. Como se menciona en el capítulo de introducción, el formato decimal soportado tiene que cumplir la condición $11 \leq \alpha + \beta \leq 15$ para tener la redundancia suficiente para prevenir una propagación de acarreo.

Aparte del RBCD, hay muchos otros códigos de 4 bits que cumplen esta condición. Sin embargo, nosotros limitamos el estudio de códigos decimales redundantes a los casos en que el bit más significativo tiene peso negativo y el resto de bits tienen un peso positivo (similar al RBCD). De esta manera, el retardo y el área de los correspondientes sumadores serán similares al caso del sumador RBCD original (otros códigos implican un alto coste hardware que no los hacen competitivos). Un estudio de distintos códigos y sumadores se presenta en el Apéndice A.

La Tabla 3.4 muestra los códigos decimales redundantes que son discutidos en esta sección. En la primera columna presentamos los pesos de cada bit y el código como función de su posición relativa. La segunda columna proporciona

UNIVERSIDAD DE MÁLAGA

Código (Peso)	Conjunto de Dígitos	Ejemplo (dígito 4)	Ejemplo (dígito -4)
-8421*	{-7,..,7}	0100	1100
-7421	{-7,..,7}	0100	1011
-7321	{-7,..,6}	0101	1011—1100
-6421	{-6,..,7}	0100	1010
-6321	{-6,..,6}	0101	1010
-6221	{-6,..,5}	0110	1010—1100
-5421	{-5,..,7}	0100	1001
-5321	{-5,..,6}	0101	1001
-4421	{-4,..,7}	0100	1000

* Estándar RBCD

Tabla 3.4: Códigos RBCD para la suma decimal multiformato

el conjunto de dígitos de cada código, y la tercera y cuarta columna muestran algunos ejemplos de cada código. El primer código de la Tabla (con peso -8421) corresponde al RBCD. Para mayor claridad, a partir de ahora nos referimos a $RBCD_{-7421}, RBCD_{-7321}$... como los códigos RBCD con peso -7421, -7321 ... , y mantenemos la notación RBCD para el código $RBCD_{-8421}$ (es decir, $RBCD = RBCD_{-8421}$).

Llamaremos olDFA$_{Mformat}$ a un sumador decimal online multiformato con dos operandos. Hay dos maneras de diseñar un sumador multiformato:

i) usando el sumador olDFA existente con una etapa de conversión a RBCD,

ii) usando sumadores *ad hoc* para los códigos específicos asociados.

En las siguientes subsecciones vamos a ver estos dos métodos y discutiremos sus ventajas e inconvenientes.

3.2.1. Multiformato mediante Etapa de Conversión

El olDFA descrito en la sección 2.3 trabaja con operandos codificados en RBCD. Una manera de diseñar un sumador decimal online multiformato es mediante el uso de un olDFA que esté precedido por una etapa de conversión. En esta etapa se convierte un código de los que se muestra en la Tabla 3.4 al código RBCD. El retardo y el área del módulo de conversión depende de las funciones de conversión específicas del código de entrada. Por ejemplo, las funciones de conversión del código RBCD$_{-5421}$ (X_i^3,x_i^2, x_i^1, x_i^0) al código RBCD ($X_i'^3$,$x_i'^2$, $x_i'^1$, $x_i'^0$) son:

UNIVERSIDAD DE MÁLAGA

$$
\begin{aligned}
X_i^{'3} &= X_i^3 \cdot (\overline{x_i^2} + \overline{x_i^1} \cdot \overline{x_i^0}) \\
x_i^{'2} &= \overline{X_i^3} \cdot x_i^2 + X_i^3 \cdot ((x_i^1 + x_i^0) \oplus x_i^2) \\
x_i^{'1} &= \overline{X_i^3} \cdot x_i^1 + X_i^3 \cdot (\overline{x_i^1 \oplus x_i^0}) \\
x_i^{'0} &= x_i^0 \oplus X_i^3
\end{aligned}
\tag{3.5}
$$

Como ejemplo de la conversión anterior, el número 1101_{-5421} es igual a 0000_{RBCD} ya que:

$$
\begin{aligned}
X_i^{'3} &= 1 \cdot (\overline{1} + \overline{0} \cdot \overline{1}) = 1 \cdot 0 = 0 \\
x_i^{'2} &= \overline{1} \cdot 1 + 1 \cdot ((0 + 1) \oplus 1) = 0 + 1 \cdot (0) = 0 \\
x_i^{'1} &= \overline{1} \cdot 0 + 1 \cdot (\overline{0 \oplus 1}) = 0 + 1 \cdot (\overline{1}) = 0 \\
x_i^{'0} &= 1 \oplus 1 = 0
\end{aligned}
\tag{3.6}
$$

Observando las ecuaciones vemos que el camino crítico de estas ecuaciones va a través de cuatro niveles lógicos[1]. Por lo tanto, para sumar números RBCD_{-5421} y RBCD, necesitamos convertir el operando codificado en RBCD_{-5421} a RBCD, y después sumar los dos operandos RBCD mediante un olDFA.

Hemos calculado las funciones de conversión para todos los códigos de la Tabla 3.4 a RBCD y se pueden encontrar en el Apéndice B. La Tabla 3.5 muestra el número de niveles lógicos necesarios para dichas funciones de conversión.

Codificación	Niveles Lógicos
RBCD	0
RBCD_{-7421}	4
RBCD_{-7321}	4
RBCD_{-6421}	3
RBCD_{-6321}	4
RBCD_{-6221}	2
RBCD_{-5421}	4
RBCD_{-5321}	3
RBCD_{-4421}	1

Tabla 3.5: Retardo de la conversión a RBCD de los códigos presentes en la tabla

La figura 3.8 muestra la arquitectura de un $\text{olDFA}_{Mformat}$ general mediante etapa conversión, donde cada entrada de 4 bits (X,Y) tiene una señal (f1,f2) que

[1] Un nivel lógico corresponde a una puerta de tres entradas; este modelo es usado en [16] y nosotros lo usamos como aproximación. Esto se corrobora con los resultados de la implementación real en la subsección 3.2.6

UNIVERSIDAD DE MÁLAGA

representa el formato del número entrante. El área y retardo del módulo `Code Conversion` depende de la codificación de los operandos de entrada, así como del número de formatos distintos soportados por cada entrada.

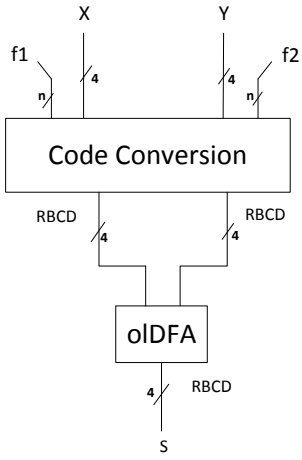


Figura 3.8: olDFA$_{Mformat}$ mediante etapa de conversión.

Por ejemplo, la arquitectura más simple de un olDFA$_{Mformat}$ mediante etapa de conversión es la que soporta dos formatos por cada entrada, una codificada en RBCD y la otra codificada en RBCD$_{-4421}$ (véase la figura 3.9). El olDFA$_{Mformat}$ mediante etapa de conversión más complejo es el que cubre todos los códigos de la Tabla 3.4 (véase la figura 3.10). En cualquier caso, el retardo online de un olDFA$_{Mformat}$ mediante etapa de conversión es $\delta_{olDFA_{Mformat}} = 1$ siendo igual que el de un olDFA. Sin embargo, el ciclo de reloj de un olDFA$_{Mformat}$ es:

$$CC_{olDFA_{Mformat}} = CC_{olDFA} + \Delta \tag{3.7}$$

donde Δ varía y depende del número de formatos soportados por las entradas del olDFA$_{Mformat}$.

Volviendo a los ejemplos anteriores, si el olDFA$_{Mformat}$ soporta el caso más simple (RBCD y RBCD$_{-4421}$), entonces $\Delta = T_{mux2-1} + 1ll$, donde T_{mux2-1} es el retardo de un multiplexor 2-1 y ll el retardo de un nivel lógico (véase la figura 3.9 y la Tabla 3.5). Si el olDFA$_{Mformat}$ soporta el caso más complejo, entonces $\Delta = T_{mux9-1} + 4ll$, donde T_{mux9-1} es el retardo de un multiplexor 9-1 (véase la figura 3.10 y la Tabla 3.5).

UNIVERSIDAD DE MÁLAGA

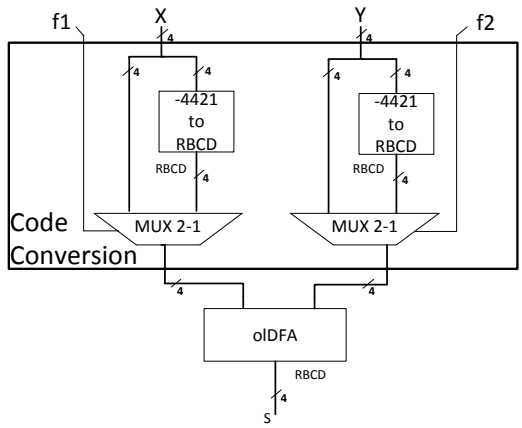


Figura 3.9: El caso más simple: olDFA$_{Mformat}$ soportando dos códigos RBCD y RBCD$_{\text{-}4421}$.

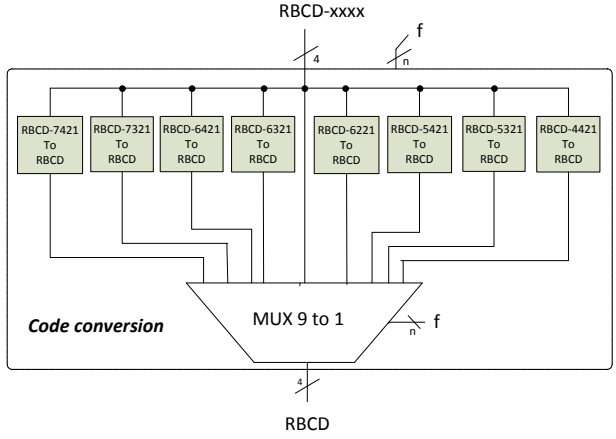


Figura 3.10: El caso más complejo: Módulo de conversión cubriendo todos los casos de la Tabla 3.4.

3.2.2. Multiformato mediante Diseño Específico

Esta subsección muestra el diseño de un olDFA$_{Mformat}$ que soporta dos códigos diferentes en ambas entradas del sumador sin un módulo de etapa de conversión. El objetivo es reducir el coste en tiempo extra debido a la etapa de conversión

UNIVERSIDAD DE MÁLAGA

de la propuesta anterior mediante el diseño de sumadores específicos para cada pareja posible de códigos de la Tabla 3.4. En la subsección 3.2.4 se discute el caso en el que se tiene más de dos códigos por entrada. A partir de ahora, nos referiremos a estos sumadores específicos como *sumadores multiformato mediante diseño específico*.

Para diseñar estos sumadores específicos con salida RBCD, seguimos basándonos en el esquema de descomposición de los operandos propuesto en [16] y usado en el diseño del olDFA (como se muestra en la figura 2.4).

La figura 3.11 muestra un ejemplo de este esquema de descomposición para el caso multiformato donde los círculos blancos representan negabits y los círculos negros representan posibits; los grupos Z_i, V_i y t_{i+1} se calculan siguiendo las ecuaciones (2.3) (presentadas en la sección 2.2).

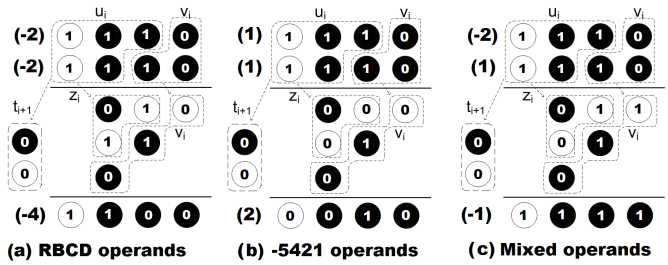
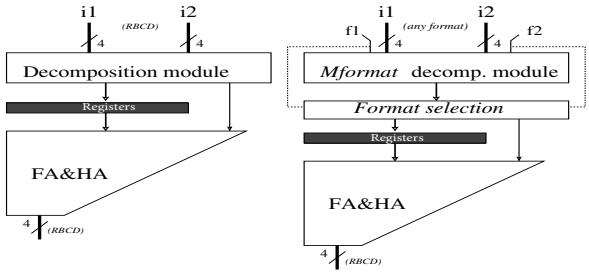


Figura 3.11: Ejemplo de esquema de descomposición para multiformato.

La figura 3.11 muestra un ejemplo para los códigos RBCD y RBCD$_{-5421}$, donde, tomando el mismo dato binario (1110 más 1110) como entrada, se interpreta de distinta manera en función del formato de los operandos. En dicho ejemplo se obtienen tres resultados diferentes mientras que se mantiene el mismo esquema de descomposición. En consecuencia, se necesitan tres submódulos diferentes ya que hay que implementar una función por cada resultado diferente.

El olDFA$_{Mformat}$ propuesto en esta subsección se basa en una modificación de la arquitectura interna del olDFA. La figura 3.12 compara el diagrama de bloques de un olDFA y del nuevo olDFA$_{Mformat}$. Se puede ver que hay un nuevo módulo para la selección de formato (`Format selection` en la figura) y un nuevo módulo de descomposición (`Mformat decomposition module`). A continuación, presentamos en detalle la arquitectura de los módulos `Mformat decomposition` y `Format selection` de la figura 3.12.b.

UNIVERSIDAD DE MÁLAGA



(a) Regular olDFA (b) olDFA$_{\text{Mformat}}$

Figura 3.12: Diagramas de bloque del olDFA y del olDFA$_{Mformat}$ mediante diseño específico.

Módulo de Descomposición

En un olDFA, se suman dos números RBCD y es el módulo de descomposición el que realiza las ecuaciones (2.3).

En el olDFA$_{Mformat}$ el nuevo módulo de descomposición es más complejo y las ecuaciones asociadas dependen de los códigos específicos soportados por las entradas. De hecho, las ecuaciones (2.3) son un caso particular cuando hay dos entradas RBCD en el nuevo olDFA$_{Mformat}$.

Consideremos un olDFA$_{Mformat}$ con entradas i1 e i2, que soporta las codificaciones con formato A y B en cada entrada, y que la señal f1 (f2) es el bit de control para seleccionar el formato de la entrada i1 (i2). Las cuatro combinaciones posibles de la entrada son las siguientes:

```
i1 i2  <-- Entradas
------
A  A
B  B   <-- Combinaciones de Codificaciones
A  B
B  A
```

Las ecuaciones para realizar la descomposición de cada combinación de códigos son diferentes. Por ello, requieren diferentes submódulos hardware para su implementación. Sin embargo, las ecuaciones para las combinaciones AB y BA son muy parecidas y pueden ser unificadas e integradas en un único submódulo hardware (con una complejidad similar a la de las combinaciones AA y BB). La figura 3.13 muestra la arquitectura general del módulo `Mformat decomposition`,

UNIVERSIDAD DE MÁLAGA

donde los tres diferentes submódulos hardware se encargan de realizar la descomposición para las combinaciones AA, BB y AB&BA. La señal $f2$ se usa en el submódulo AB&BA para identificar la combinación de códigos de la entrada (que puede ser AB o BA).

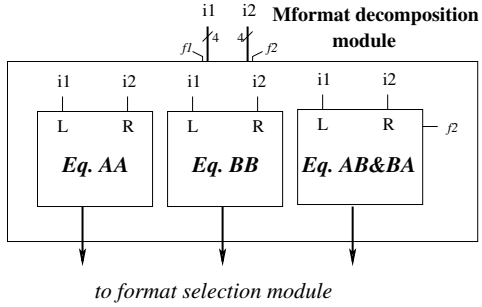


Figura 3.13: Módulo de descomposición de olDFA$_{Mformat}$ mediante diseño específico.

Teniendo en cuenta los nueve códigos de la Tabla 3.4, hay 36 parejas diferentes de códigos, por lo que hay 36 diseños de olDFA$_{Mformat}$ diferentes. Se han diseñado e implementado algunos de esos olDFA$_{Mformat}$. Aunque no se han diseñado los 36 sumadores posibles, podemos deducir, basándonos en nuestro estudio, que el área y el retardo son similares para todos ellos. De hecho, en el Apéndice C, analizamos cómo construir un olDFA$_{Mformat}$ específico que soporte una entrada RBCD y cualquier entrada codificada con cualquier código de la Tabla 3.4.

A continuación y como ejemplo, presentamos un diseño concreto donde se usan los códigos RBCD y RBCD$_{-5421}$ (RBCD$_{-5421}$ se usa en los multiplicadores decimales presentados en [36] con el objetivo de mejorar el área y la latencia de la reducción del producto parcial). La Tabla 3.6 muestra la equivalencia entre los códigos RBCD y RBCD$_{-5421}$.

El olDFA$_{Mformat}$ propuesto puede sumar dos operandos, ambos codificados en RBCD o ambos codificados en RBCD$_{-5421}$ o uno en RBCD y el otro en RBCD$_{-5421}$. A continuación estudiaremos los tres casos por separado.

- **Ambos operandos están codificados en RBCD** (caso de la figura 3.11.a y Ecuación AA en la figura 3.13). Éste es el caso del olDFA, donde las ecuaciones (2.3) se implementan en el módulo de descomposición. El camino crítico de estas ecuaciones es de 4 niveles lógicos (debido al cálculo de z_i^2).

UNIVERSIDAD DE MÁLAGA

	RBCD	RBCD_{-5421}
-7	1001	-
-6	1010	-
-5	1011	1000
-4	1100	1001
-3	1101	10101
-2	1110	1011
-1	1111	1100
0	0000	0000—1101
1	0001	0001—1110
2	0010	0010—1111
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

Tabla 3.6: Comparación de los códigos RBCD y RBCD_{-5421}

- **Ambos operandos están codificados en RBCD_{-5421}** (caso de la figura 3.11.b y Ecuación BB en la figura 3.13). Las ecuaciones implementadas en el módulo de descomposición son las siguientes:

$$
\begin{aligned}
t^0_{5421_{i+1}} &= (\overline{X^3_i} \cdot \overline{Y^3_i}) + (X^3_i \oplus Y^3_i) \cdot x^2_i \cdot y^2_i \\
T^0_{5421_{i+1}} &= \overline{(X^3_i \oplus Y^3_i)} \cdot (\overline{x^2_i} \cdot \overline{y^2_i}) \\
Z^2_{5421_i} &= \overline{X^3_i} \cdot \overline{Y^3_i} \cdot (\overline{\overline{x^2_i} \cdot x^1_i \cdot \overline{y^2_i}}) \\
&\quad + X^3_i \oplus Y^3_i \cdot (x^2_i \cdot \overline{x^1_i} + \overline{x^1_i} \cdot \overline{y^2_i}) \\
&\quad + X^3_i \cdot Y^3_i \cdot (x^2_i \oplus y^2_i) \\
z^2_{5421_i} &= \overline{X^3_i} \cdot \overline{Y^3_i} \cdot (\overline{x^2_i \oplus y^2_i}) \\
&\quad + X^3_i \cdot Y^3_i \cdot (\overline{x^2_i} \cdot x^1_i \cdot \overline{y^2_i}) \\
&\quad + (X^3_i \oplus Y^3_i) \cdot (x^2_i \cdot \overline{y^2_i} + \overline{x^2_i} \cdot x^1_i \cdot y^2_i) \\
Z^1_{5421_i} &= \overline{(X^3_i \oplus Y^3_i)} \cdot (\overline{x^2_i} \cdot (x^1_i \oplus y^2_i) + x^2_i \cdot \overline{x^1_i}) \\
&\quad + (X^3_i \oplus Y^3_i) \cdot (x^2_i (x^1_i \oplus y^2_i) + \overline{x^2_i} \cdot x^1_i) \\
V^0_{5421_i} &= (X^3_i \oplus Y^3_i) \oplus (x^0_i \oplus y^0_i) \\
v^1_{5421_i} &= \overline{(X^3_i \oplus Y^3_i)} \cdot (y^1_i \oplus (y^0_i + x^0_i)) \\
&\quad + (X^3_i \oplus Y^3_i) \cdot (y^1_i \oplus (x^0_i \cdot y^0_i)) \\
v^2_{5421_i} &= \overline{(X^3_i \oplus Y^3_i)} \cdot (y^1_i \cdot (y^0_i + x^0_i))
\end{aligned}
\tag{3.8}
$$

UNIVERSIDAD DE MÁLAGA

$$+y_i^1 \cdot x_i^0 \cdot y_i^0$$

El camino crítico es de 4 niveles lógicos (debido al cálculo de $Z^1_{5421_i}$). El retardo para dos entradas RBCD$_{-5421}$ es el mismo que para dos entradas RBCD.

- **Un operando está codificado en RBCD y el otro está codificado en RBCD$_{-5421}$** (caso de la figura 3.11.c y Ecuación AB&BA en la figura 3.13). En este caso, la descomposición se realiza en dos pasos:

  i) computación en paralelo de las ecuaciones (3.8) y de las siguientes ecuaciones:

$$\begin{aligned}
t^0_{neg_{i+1}} &= (X_i^3 \cdot \overline{Y_i^3} \cdot f2 + \overline{X_i^3} \cdot Y_i^3 \cdot \overline{f2}) \cdot x_i^2 \cdot y_i^2 \\
T^0_{neg_{i+1}} &= (X_i^3 \cdot \overline{Y_i^3} \cdot f2 + \overline{X_i^3} \cdot Y_i^3 \cdot \overline{f2}) \cdot \overline{x_i^2} \cdot \overline{x_i^1} \cdot \overline{y_i^2} \\
&\quad + X_i^3 \cdot Y_i^3 \cdot \overline{x_i^1} \cdot (x_i^2 \oplus y_i^2) \\
Z^2_{neg_i} &= (X_i^3 \cdot \overline{Y_i^3} \cdot f2 + \overline{X_i^3} \cdot Y_i^3 \cdot \overline{f2}) \cdot (\overline{x_i^2} + x_i^2 \cdot y_i^2) \\
&\quad + X_i^3 \cdot Y_i^3 \cdot \overline{x_i^1} \cdot (x_i^2 + y_i^2) \\
z^2_{neg_i} &= X_i^3 \cdot Y_i^3 \cdot (\overline{y_i^2} \cdot (x_i^2 \oplus x_i^1)) \\
&\quad + (X_i^3 \cdot \overline{Y_i^3} \cdot f2 + \overline{X_i^3} \cdot Y_i^3 \cdot \overline{f2}) \cdot (x_i^2 \cdot x_i^1 + \overline{x_i^1} \cdot \overline{y_i^2}) \\
Z^1_{neg_i} &= X_i^3 \cdot Y_i^3 \cdot \overline{(x_i^2 \cdot \overline{x_i^1} \cdot \overline{y_i^2})} \\
&\quad + (X_i^3 \cdot \overline{Y_i^3} \cdot f2 + \overline{X_i^3} \cdot Y_i^3 \cdot \overline{f2}) \cdot \overline{x_i^1} \cdot \overline{(x_i^2 \oplus y_i^2)} \\
V^0_{neg_i} &= X_i^3 \cdot f2 + Y_i^3 \cdot \overline{f2} \\
v^1_{neg_i} &= (X_i^3 \cdot f2 + Y_i^3 \cdot \overline{f2}) \cdot (y_i^0 \oplus x_i^0) \\
v^2_{neg_i} &= (X_i^3 \cdot f2 + Y_i^3 \cdot \overline{f2}) \cdot y_i^1 \cdot (y_i^0 \oplus x_i^0)
\end{aligned} \tag{3.9}$$

  y ii) operación XOR entre ellas, esto es,

$$\begin{aligned}
t^0_{mix_{i+1}} &= t^0_{neg_{i+1}} \oplus t^0_{5421_{i+1}} \\
T^0_{mix_{i+1}} &= T^0_{neg_{i+1}} \oplus T^0_{5421_{i+1}} \\
Z^2_{mix_i} &= Z^2_{neg_i} \oplus Z^2_{5421_i} \\
z^2_{mix_i} &= z^2_{neg_i} \oplus z^2_{5421_i}
\end{aligned}$$

UNIVERSIDAD DE MÁLAGA

$$
\begin{aligned}
Z^1_{mix_i} &= Z^1_{neg_i} \oplus Z^1_{5421_i} \\
V^0_{mix_i} &= V^0_{neg_i} \oplus V^0_{5421_i} \\
v^1_{mix_i} &= v^1_{neg_i} \oplus v^1_{5421_i} \\
v^2_{mix_i} &= v^2_{neg_i} \oplus v^2_{5421_i}
\end{aligned} \tag{3.10}
$$

Hay que tener en cuenta que las ecuaciones (3.9) incluyen la señal $f2$ la cual asigna la codificación a la entrada correcta. Las ecuaciones (3.8) y (3.9) se implementan en el módulo `Mformat decomposition`. Por otro lado, las ecuaciones (3.10) se implementan fuera de este módulo, concretamente, en el módulo `Format selection` como se explica en la siguiente subsección.

Módulo de Selección de Formato

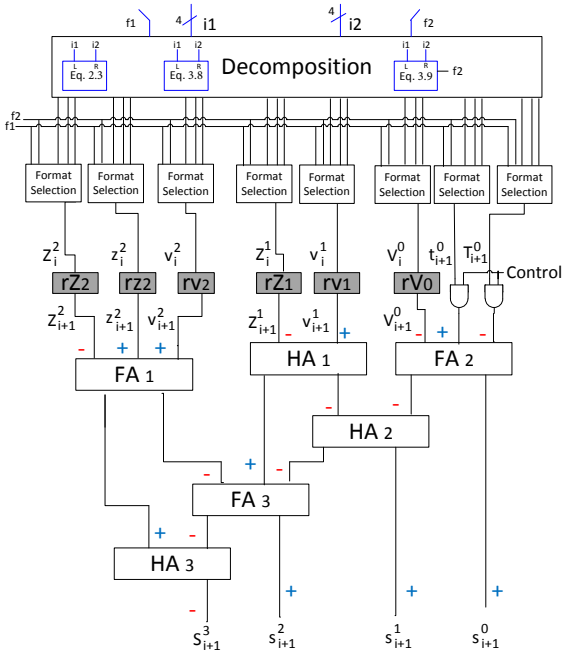


Figura 3.14: Arquitectura general del olDFA$_{Mformat}$ mediante diseño específico.

La figura 3.14 muestra en detalle la arquitectura general del olDFA$_{Mformat}$ mediante diseño específico. Dado que el retardo de las ecuaciones (2.3), (3.8), y

UNIVERSIDAD DE MÁLAGA

(3.9) es de cuatro niveles lógicos y trabajan en paralelo, el retardo del módulo de descomposición del olDFA$_{Mformat}$ es el mismo que el del olDFA (que sólo implementa las ecuaciones (2.3)). Por lo tanto, desde el punto de vista del tiempo de computación, el coste extra del soporte multiformato comparado con el de un olDFA normal es debido solamente al módulo `Format selection` de la figura 3.14.

La figura 3.15 muestra la estructura interna del módulo `Format selection` para la señal Z^1 (se requiere el mismo hardware para las restantes señales $t^0, T^0, Z^2, ...$, de la figura 3.14). Este módulo es el encargado de ejecutar las ecuaciones (3.10). Las señales $f1$ y $f2$ se utilizan para seleccionar los códigos de los operandos como se muestra en la figura 3.15.

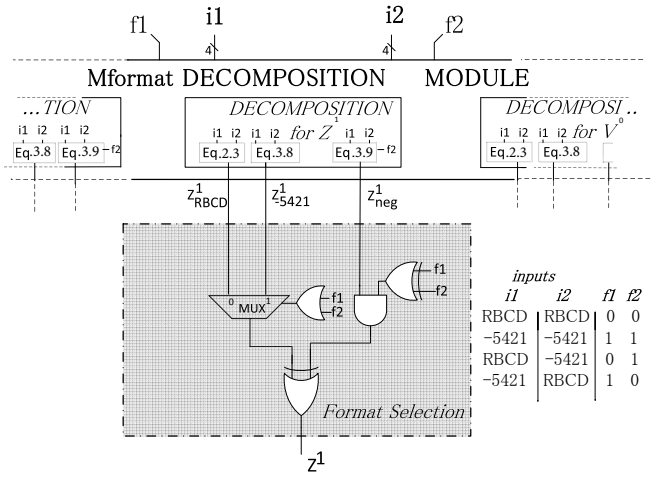


Figura 3.15: Selección de formato para Z^1 (HW similar para las restantes señales que llegan del módulo `Mformat decomposition`).

De acuerdo con la figura 3.15, el retardo del módulo `Format selection` es el de un multiplexor 2-1 más una puerta XOR. Obsérvese que las puertas lógicas OR y XOR con las entradas $f1, f2$ trabajan en paralelo con el módulo `Mformat decomposition` y, por lo tanto, no están en el camino crítico del módulo `Format selection`. Luego el retardo extra para convertir el olDFA en un sumador online multiformato es equivalente al del módulo `Format selection`.

UNIVERSIDAD DE MÁLAGA

3.2.3. Segmentación del olDFA$_{Mformat}$

Segmentación del olDFA$_{Mformat}$ mediante Etapa de Conversión

Para segmentar el olDFA$_{Mformat}$ mediante etapa de conversión con etapas balanceadas, tenemos que tener en cuenta la complejidad del módulo de conversión. Si se tiene el caso en el que sólo se trata con dos formatos, dado que los retardos de las conversiones de códigos de la Tabla 3.5 van desde 1 a 4 niveles lógicos y el módulo de descomposición tiene un retardo de 4 niveles lógicos, la mejor opción es la de insertar los registros de segmentación justo a la salida del módulo de etapa de conversión y reemplazar el olDFA por un olDFA$_p$. La arquitectura resultante se muestra en la figura 3.16. Esta arquitectura segmentada tiene el mismo ciclo de reloj que el de un olDFA$_p$ y el retardo online es $\delta_{olDFA_p}+1$ (debido a la fila de registros en la salida del módulo `Code Conversion`).

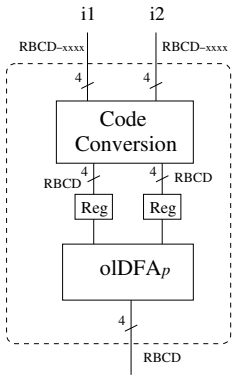


Figura 3.16: Arquitectura segmentada del olDFA$_{Mformat}$ mediante etapa de conversión.

En el caso de tener un módulo de conversión más complejo (el de un olDFA$_{Mformat}$ que soporte más de dos formatos en la entrada), y querer mantener un ciclo de reloj similar al de un olDFA$_p$, se podría realizar un ajuste insertando un nuevo nivel de registros de segmentación justo antes del multiplexor final del módulo de etapa de conversión (véase la figura 3.10).

Segmentación del olDFA$_{Mformat}$ mediante Diseño Específico

Para segmentar el olDFA$_{Mformat}$ mediante diseño específico, tenemos que tener en cuenta que el retardo del módulo de descomposición es de 4 niveles

UNIVERSIDAD DE MÁLAGA

lógicos y el retardo del módulo `Format selection` es menor que 4. Por lo tanto, para mantener un ciclo de reloj similar al del olDFA$_p$, insertamos registros de segmentación entre el módulo de descomposición y el módulo `Format selection`, como se muestra en la figura 3.17. En este caso, el retardo online de la arquitectura segmentada es δ_{olDFA_p}, es decir, tiene el mismo retardo online que un olDFA$_p$.

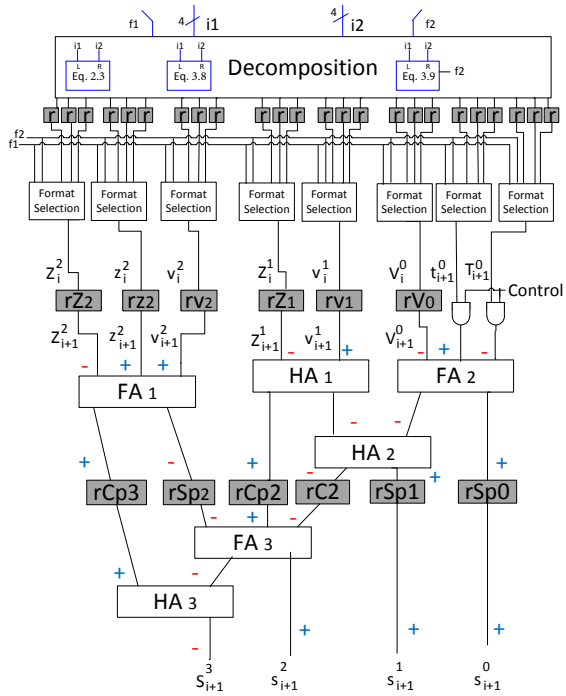


Figura 3.17: Arquitectura segmentada del olDFA$_{Mformat}$ mediante diseño específico.

3.2.4. Comparación entre olDFA$_{Mformat}$ mediante Etapa de Conversión y mediante Diseño Específico

En esta sección se discuten las ventajas e inconvenientes de los diseños propuestos del olDFA$_{Mformat}$. El primer caso es el del olDFA$_{Mformat}$ soportando dos formatos diferentes. El coste extra del olDFA$_{Mformat}$ mediante etapa de conversión es debido al aumento del retardo por la etapa de conversión y un multiplexor 2-1 (para seleccionar uno de los dos formatos en la entrada) mostrado en

UNIVERSIDAD DE MÁLAGA

la figura 3.9. Por otro lado, el coste extra de un olDFA$_{Mformat}$ mediante diseño específico es igual al retardo de un multiplexor 2-1 más el retardo de una puerta XOR (véase la figura 3.15). Por ello, desde el punto de vista de rendimiento en tiempo, el olDFA$_{Mformat}$ mediante diseño específico es el mejor, como era de esperar. Sin embargo, el coste hardware de la primera implementación es menor que el de la segunda, debido a la complejidad del módulo de descomposición del olDFA$_{Mformat}$ mediante diseño específico.

Para el caso de un olDFA$_{Mformat}$ con tres o más formatos soportados en cada entrada, el módulo de descomposición correspondiente a un olDFA$_{Mformat}$ mediante diseño específico se hace extremadamente complejo, lo que lo hace poco práctico. Por ello, para estos casos, el olDFA$_{Mformat}$ mediante etapa de conversión es la alternativa más razonable ya que el módulo de etapa de conversión tiene un reducido coste hardware (véase las funciones de conversión en el Apéndice B), y el tiempo extra es debido sólo al multiplexor final (incremento logarítmico). Esto se debe a que todos los elementos de la función de conversión trabajan en paralelo dentro del módulo de etapa de conversión. En conclusión, para dos operandos el diseño específico es una buena elección mientras que para tres o más operandos el diseño con etapa de conversión parece más apropiado.

3.2.5. Suma Decimal Online Multioperando y Multiformato

Si una aplicación necesita tratar con multioperandos con formatos diferentes (multiformato), es posible diseñar un árbol de sumadores siguiendo las estrategias de diseño presentadas en la sección 3.1. El primer nivel de los árboles tiene que estar compuesto por sumadores decimales online que soporten operandos multiformato (olDFA$_{Mformat}$), mientras que los restantes niveles estarán compuestos por olDFAs normales. Esto se debe al hecho de que la salida del olDFA$_{Mformat}$ es un número codificado en RBCD, por lo que el segundo y sucesivos niveles del árbol trabajarán con números RBCD, y por tanto las sumas de los operandos pueden ser realizadas por olDFAs normales. La figura 3.18 muestra una arquitectura general de un árbol multioperando y multiformato.

La complejidad de los diferentes olDFA$_{Mformat}$ del primer nivel del árbol depende del formato soportado por cada aplicación. El análisis de las diferentes configuraciones, tiempo y retardo online presentado en la sección 3.1 es válido para los árboles decimales multioperando y multiformato.

Para un árbol segmentado, todas las alternativas propuestas en la sección 3.1 pueden ser usadas reemplazando las unidades olDFA y olDFA$_{Mformat}$ de la figura

UNIVERSIDAD DE MÁLAGA

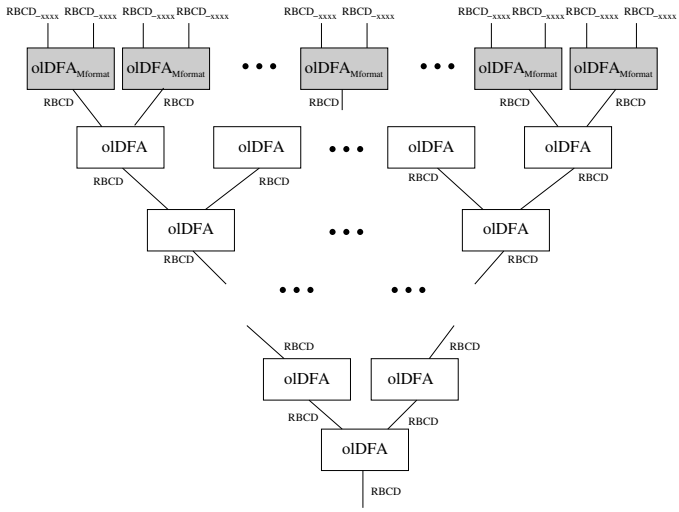


Figura 3.18: Arquitectura de un árbol multioperando y multiformato.

3.18 por su correspondiente versión segmentada.

3.2.6. Resultados Experimentales

Los diseños presentados en esta sección han sido implementados en Verilog, simulados usando ModelSim 6.0, y sintetizados usando Synopsys Design Compiler y la liberia TSMC 65nm en la cual una unidad de celda tiene un área igual a 1 μm^2.

Ambos olDFA$_{Mformat}$s, mediante etapa de conversión y mediante diseño específico, fueron verificados para las 225 combinaciones diferentes de entrada (variaciones con repetición de 15 elementos tomados de dos en dos). A la hora de la sintetización de los diseños, configuramos dos escenarios de simulación diferentes activando o desactivando el atributo *dont_touch* en los diseños. En el primer escenario, que denotamos como S1, activamos el atributo *dont_touch* para evitar la modificación de la estructura de los olDFAs, olDFA$_p$s y olDFA$_{Mformat}$s en los árboles. De esta manera, la estructura se mantiene y puede ser analizada desde un punto de vista teórico/analítico. En el segundo escenario de simulación, que denotamos como S2, desactivamos el atributo *dont_touch* para permitir a la herramienta Synopsys DC realizar una optimización balanceada entre área y retardo de las arquitecturas.

UNIVERSIDAD DE MÁLAGA

3.2.7. Rendimiento del olDFA$_{Mformat}$

Las Tablas 3.7 y 3.8 y las Tablas 3.9 y 3.10 muestran el área, retardo, retardo online (δ), intervalo de iniciación (I.I.), y el throughput (THR.) de los olDFA$_{Mformat}$s descritos en la sección 3.2. Más específicamente, estas tablas muestran los resultados de los olDFA$_{Mformat}$s más representativos: el más simple mediante etapa de conversión (que soporta operandos RBCD y RBCD$_{\text{-}4421}$, denotado como RBCD+
RBCD$_{\text{-}4421}$ CodeConv), el más complejo mediante etapa de conversión (que soporta los nueve códigos de la Tabla 3.5 y denotado como 9 codes CodeConv), y dos olDFA$_{Mformat}$s con operandos RBCD y RBCD$_{\text{-}5421}$, uno diseñado siguiendo el esquema de etapa de conversión (denotado como RBCD+RBCD$_{\text{-}5421}$ CodeConv) y otro siguiendo el esquema de diseño específico (denotado como RBCD+RBCD$_{\text{-}5421}$ SpecD). Para poder hacer comparaciones y analizar los diseños del olDFA$_{Mformat}$, también realizamos la sintetización de los olDFA y olDFA$_p$ bajo los escenarios de simulación S1 y S2. Los resultados se muestran en las Tablas 3.11 y 3.12.

olDFA$_{Mformat}$	No Segmentado				
	retardo ns	área $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operac. por s
RBCD+RBCD$_{\text{-}4421}$ CodeConv	0.387	530	1	6.58	161
9 codes CodeConv	0.493	1999	1	8.38	126
RBCD+RBCD$_{\text{-}5421}$ CodeConv	0.412	747	1	7.00	151
RBCD+RBCD$_{\text{-}5421}$ SpecD	0.342	1046	1	5.81	182

I.I.=Intervalo de Iniciación, THR.=Rendimiento, δ=retardo online,
CodeConv= Conversión de Código, SpecD= Diseño Específico

Tabla 3.7: Resultados de olDFA$_{Mformat}$ no segmentado en el primer escenario de simulación (S1)

olDFA$_{Mformat}$	Segmentado				
	retardo ns	área $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operac. por s
RBCD+RBCD$_{\text{-}4421}$ CodeConv	0.218	649	4	4.36	286
9 codes CodeConv	0.236	2073	4	4.72	264
RBCD+RBCD$_{\text{-}5421}$ CodeConv	0.218	797	4	4.36	286
RBCD+RBCD$_{\text{-}5421}$ SpecD	0.21	1124	3	3.99	297

I.I.=Inter. de Iniciación, THR.=Rendimiento, δ=retardo online,
CodeConv= Conversión de Código, SpecD= Diseño Específico

Tabla 3.8: Resultados de olDFA$_{Mformat}$ segmentado en el primer escenario de simulación (S1)

UNIVERSIDAD DE MÁLAGA

La Tablas 3.7 y 3.8 muestran los resultados de los diseños bajo S1. Como puede observarse los resultados presentados en las tablas son consistentes con la teoría, esto es, el olDFA$_{Mformat}$ mediante etapa de conversión con 9 códigos (9 codes CodeConv) es el que peor rendimiento tiene, mientras que RBCD+RBCD$_{\text{-}5421}$ SpecD es el mejor en términos de retardo, throughput e intervalo de iniciación. De hecho, el coste extra en el retardo de RBCD+RBCD$_{\text{-}5421}$ SpecD comparado con el retardo del olDFA bajo S1 (véanse las Tablas 3.11 y 3.12) es debido al módulo de selección de formato (véase la figura 3.15).

Como era de esperar el área del RBCD+RBCD$_{\text{-}5421}$ SpecD es mayor que el área del RBCD+RBCD$_{\text{-}5421}$ CodeConv (debido a la complejidad del módulo de descomposición), lo cual es también consistente con la teoría.

Con respecto a la versión segmentada de los olDFA$_{Mformat}$s, la tendencia en su rendimiento es similar al de las versiones no segmentadas. Los resultados muestran que se consigue una optimización del 39 % en el retardo de RBCD+RBCD$_{\text{-}5421}$ SpecD a coste de un incremento del área del 8 % con respecto a la versión no segmentada. Hay que tener en cuenta que el retardo de las versiones segmentadas del olDFA$_{Mformat}$s presentadas es prácticamente el mismo (excepto para el diseño 9 codes CodeConv) y que concuerda con el retardo del olDFA$_p$, lo cual es nuevamente consistente con la teoría (véase la Tabla 3.12). Esto es debido a que el camino crítico de los olDFA$_{Mformat}$s segmentados está en el módulo de descomposición de sus olDFA$_p$s.

Los resultados obtenidos bajo S2 se muestran en las Tablas 3.9 y 3.10. Como se ha mencionado, el objetivo de este escenario es balancear área y retardo en los diseños. Bajo este escenario, RBCD+RBCD$_{\text{-}4421}$ CodeConv logra el mejor resultado en términos de área y retardo para ambas versiones segmentadas y no segmentadas. De hecho, los retardos de ambos diseños de RBCD+RBCD$_{\text{-}4421}$ CodeConv segmentados y no segmentados son prácticamente igual que el retardo

olDFA$_{Mformat}$	No segmentado				
	retardo ns	área $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operac. por s
RBCD+RBCD$_{\text{-}4421}$ CodeConv	0.309	522	1	5.25	202
9 codes CodeConv	0.43	1366	1	7.31	145
RBCD+RBCD$_{\text{-}5421}$ CodeConv	0.34	610	1	5.78	183
RBCD+RBCD$_{\text{-}5421}$ SpecD	0.341	896	1	5.79	183

I.I.=Intervalo de Iniciación, THR.=Rendimiento, δ=retardo online,
CodeConv= Conversión de Código, SpecD= Diseño Específico

Tabla 3.9: Resultados de olDFA$_{Mformat}$ no segmentado en el segundo escenario de simulación (S2)

UNIVERSIDAD DE MÁLAGA

olDFA$_{Mformat}$	Segmentado				
	retardo ns	área $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operac. por s
RBCD+RBCD$_{-4421}$ CodeConv	0.247	520	4	4.94	253
9 codes CodeConv	0.273	1117	4	5.46	228
RBCD+RBCD$_{-5421}$ CodeConv	0.264	510	4	5.28	236
RBCD+RBCD$_{-5421}$ SpecD	0.271	990	3	5.15	230

I.I.=Intervalo de Iniciación, THR.=Rendimiento, δ=retardo online,
CodeConv= Conversión de Código, SpecD= Diseño Específico

Tabla 3.10: Resultados de olDFA$_{Mformat}$ segmentado en el segundo escenario de simulación (S2)

del olDFA y olDFA$_p$ bajo S2 (véase las Tablas 3.11 y 3.12).

Aunque no hay ninguna otra propuesta de sumador decimal online multiformato y multioperando en la literatura, con fines de comparación, se ha sintetizado el sumador RBCD paralelo presentado en [16]. Específicamente, se ha sintetizado dos sumadores RBCD paralelos: un sumador RBCD de 16 dígitos y un sumador RBCD de 32 dígitos. El retardo y área de un sumador RBCD paralelo de 16 dígitos son 0.365 ns y 5671 μm^2, mientras que el retardo y área de un sumador RBCD paralelo de 32 dígitos son 0.42 ns y 9348 μm^2. Comparando los resultados con los obtenidos por el sumador RBCD+RBCD$_{-5421}$ SpecD (véanse las Tablas 3.9 y 3.10), concluimos que nuestra propuesta online necesita un 80 % (90 %) menos de área que el sumador paralelo de 16 dígitos (32 dígitos), con un tiempo similar para obtener el dígito más significativo. Como conclusión, estos diseños son consistentes con lo esperado para un sistema online: retardo similar para obtener el dígito más significativo y una importante reducción del coste hardware.

Escenarios de Simulación	olDFA (No Segmentado)				
	retardo ns	área $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s
S1	0.297	479	1	5.05	210
S2	0.28	486	1	4.76	223

I.I.=Intervalo de Iniciación, THR.=Throughput, δ=retardo online,
CodeConv= Conversión de Código, SpecD= Diseño Específico

Tabla 3.11: Resultados de olDFA en ambos escenarios de simulación (S1 y S2)

UNIVERSIDAD DE MÁLAGA

Escenarios de Simulación	olDFA$_p$ (Segmentado)				
	retardo ns	área $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s
S1	0.218	549	3	4.14	286
S2	0.252	439	3	4.78	248

I.I.=Intervalo de Iniciación, THR.=Rendimiento, δ=retardo online,
CodeConv= Conversión de Código, SpecD= Diseño Específico

Tabla 3.12: Resultados de olDFA$_p$ en ambos escenarios de simulación (S1 y S2)

3.2.8. Rendimiento de los Árboles de Sumadores Multiformato y Multioperando

Al igual que en la subsección previa, los resultados que se muestran a continuación también se han obtenido bajo los escenarios de simulación S1 y S2.

Cabe recordar que los olDFA$_{Mformat}$s (ya sea por diseño específico o etapa de conversión) son colocados en el primer nivel de los árboles de sumadores (véase la figura 3.18) y los restantes niveles están compuestos de olDFAs (o olDFA$_p$s en los árboles segmentados).

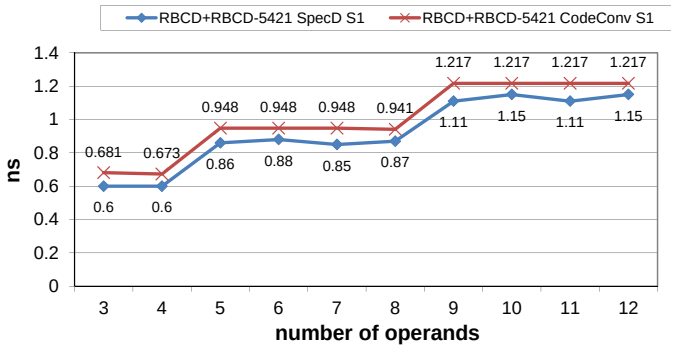
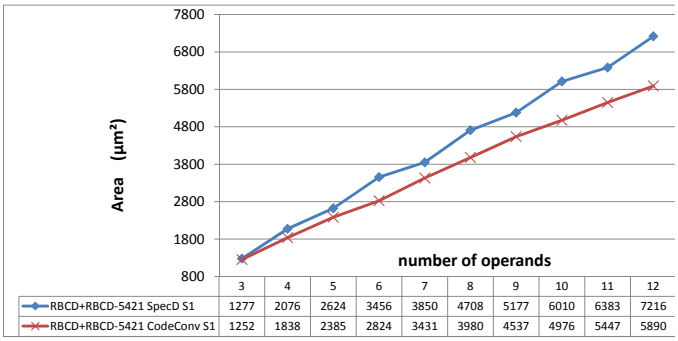


Figura 3.19: Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión no segmentada).

La figura 3.19 y la figura 3.20 muestran el retardo y área de los árboles de sumadores multiformato y multioperando no segmentados bajo S1. Como se esperaba, el retardo de los árboles no segmentados con olDFA$_{Mformat}$s mediante diseño específico es menor (entre un 4 %-10 %) que el retardo de los árboles con olDFA$_{Mformat}$s mediante etapa de conversión. Igualmente, los resultados obte-

UNIVERSIDAD DE MÁLAGA



	3	4	5	6	7	8	9	10	11	12
RBCD+RBCD-5421 SpecD S1	1277	2076	2624	3456	3850	4708	5177	6010	6383	7216
RBCD+RBCD-5421 CodeConv S1	1252	1838	2385	2824	3431	3980	4537	4976	5447	5890

Figura 3.20: Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión no segmentada).

nidos con respecto al área son los esperados, es decir, el área de los árboles con olDFA$_{Mformat}$s mediante diseño específico es mayor (entre 15 %-23 %) que el área de los árboles con olDFA$_{Mformat}$s mediante etapa de conversión. Esto es debido al número de ecuaciones implementadas en el módulo de descomposición de los olDFA$_{Mformat}$s mediante diseño específico.

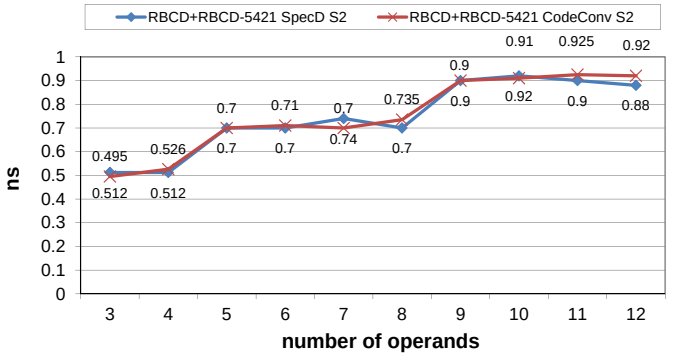
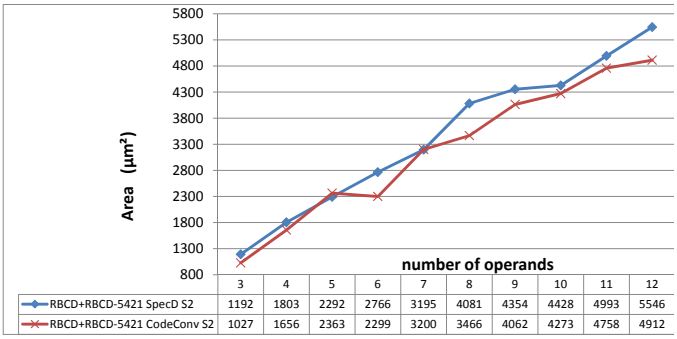


Figura 3.21: Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión no segmentada).

La figura 3.21 y la figura 3.22 muestran los resultados obtenidos usando S2 para las arquitecturas no segmentadas. El retardo y área son prácticamente iguales para ambos esquemas (diseño específico y etapa de conversión). En este caso,

UNIVERSIDAD DE MÁLAGA

la herramienta no sólo realiza un balanceo entre área y retardo, sino que también reduce el retardo y el área comparado con los datos obtenidos bajo S1 (véase la subsección 3.2.9).



	3	4	5	6	7	8	9	10	11	12
RBCD+RBCD-5421 SpecD S2	1192	1803	2292	2766	3195	4081	4354	4428	4993	5546
RBCD+RBCD-5421 CodeConv S2	1027	1656	2363	2299	3200	3466	4062	4273	4758	4912

Figura 3.22: Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión no segmentada).

La figuras 3.23, 3.24, 3.25 y 3.26 muestran el retardo y área de los árboles segmentados para los escenarios de simulación S1 y S2 respectivamente. Bajo S1, el retardo de los árboles segmentados es igual para ambos esquemas (figura 3.23). La razón de esto es que el camino crítico está en el mismo lugar: el módulo de descomposición de los olDFA$_p$s colocados en el segundo nivel de los árboles.

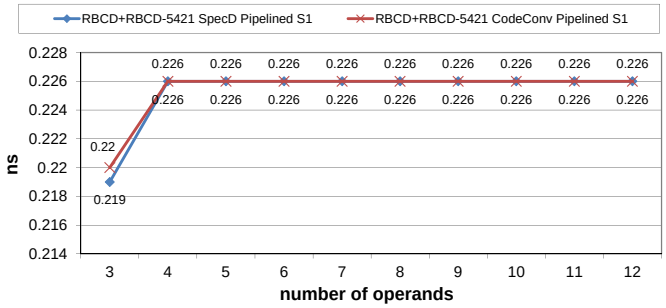
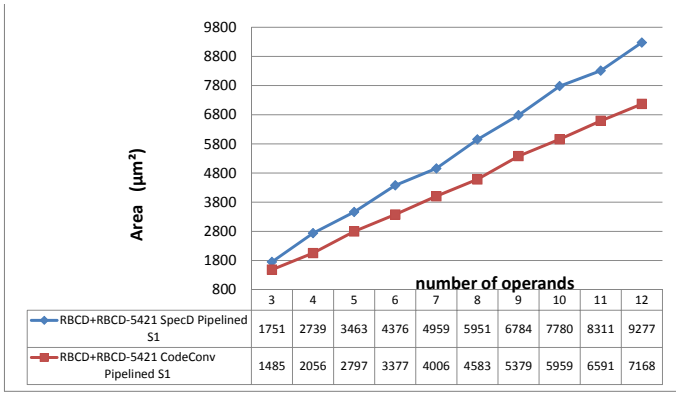


Figura 3.23: Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión segmentada).

UNIVERSIDAD DE MÁLAGA



	3	4	5	6	7	8	9	10	11	12
RBCD+RBCD-5421 SpecD Pipelined S1	1751	2739	3463	4376	4959	5951	6784	7780	8311	9277
RBCD+RBCD-5421 CodeConv Pipelined S1	1485	2056	2797	3377	4006	4583	5379	5959	6591	7168

Figura 3.24: Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 1 (Versión segmentada).

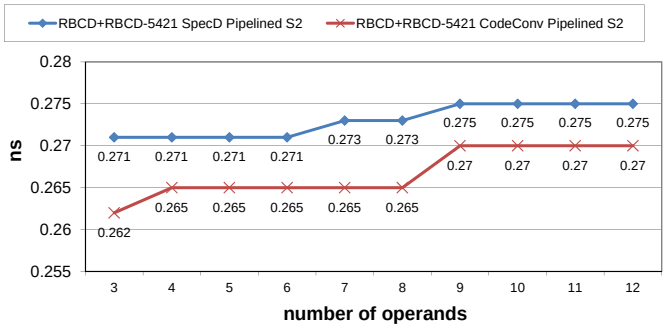
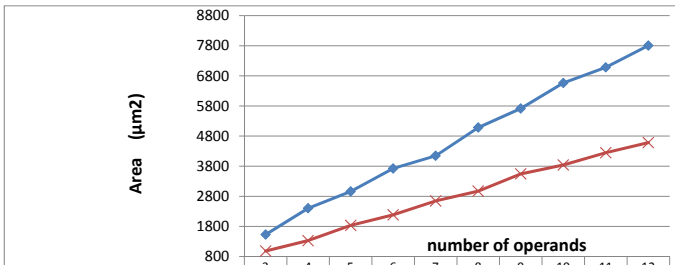


Figura 3.25: Retardo de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión segmentada).

De hecho, el retardo es prácticamente igual que el de olDFA$_p$ bajo S1 (véanse las Tablas 3.11 y 3.12). Sin embargo, el retardo online del olDFA$_{Mformat}$ mediante diseño específico (SpecD) es $\delta = 3$, es decir, uno menos que el del olDFA$_{Mformat}$ mediante etapa de conversión (CodeConv) que es ($\delta = 4$. Por lo tanto es obvio que el rendimiento, en términos de throughput, sea mejor en los casos de los árboles de sumadores segmentados mediante diseño específico. El área de los árboles de sumadores segmentados RBCD+RBCD$_{-5421}$ SpecD es entre un 17 %-33 % mayor que el de los árboles de sumadores segmentados RBCD+RBCD$_{-5421}$ CodeConv (figura 3.24) debido al área extra de los módulos de descomposición

UNIVERSIDAD DE MÁLAGA

de los olDFA$_{Mformat}$s.



	3	4	5	6	7	8	9	10	11	12
RBCD+RBCD-5421 SpecD Pipelined S2	1529	2403	2959	3724	4145	5082	5716	6562	7082	7806
RBCD+RBCD-5421 CodeConv Pipelined S2	984	1329	1833	2185	2646	2973	3543	3839	4247	4584

Figura 3.26: Área de árboles de sumadores basados en RBCD+RBCD$_{-5421}$ mediante SpecD y CodeConv bajo el escenario de simulación 2 (Versión segmentada).

Bajo S2, la diferencia en retardo en ambos esquemas es entre un 2 %-3 % (véase la figura 3.25) mientras que la diferencia en área es entre un 55 %-80 % (véase la figura 3.26).

3.2.9. Comparación entre Escenario 1 y Escenario 2 (S1 y S2)

Como se ha mencionado anteriormente, bajo S1 la herramienta mantiene las estructuras y los resultados pueden ser analizados desde un punto de vista teórico/analítico. Por otro lado, bajo S2 la herramienta balancea el retardo y el área en las arquitecturas. Esta subsección analiza la tasa de aumento/disminución (en %) del retardo y área obtenido bajo S2 comparado con el resultado obtenido bajo S1. La operación realizada para cada parámetro es (S1-S2)/S1, que significa que un valor positivo corresponde con una disminución y un valor negativo corresponde a un incremento.

Las figuras 3.27 y 3.28 muestran una comparación de los resultados obtenidos para RBCD+RBCD$_{-5421}$ SpecD y RBCD+RBCD$_{-5421}$ CodeConv bajo ambos escenarios. Se puede ver que el retardo y el área obtenidos bajo S2 se reducen para las arquitecturas no segmentadas. El retardo es prácticamente igual para RBCD+RBCD$_{-5421}$ SpecD, mientras que el área se ha reducido alrededor de un 14 %. Sin embargo, para las arquitecturas segmentadas, el retardo bajo S2 se ha incrementado y el área se ha reducido. En particular, el retardo del

UNIVERSIDAD DE MÁLAGA

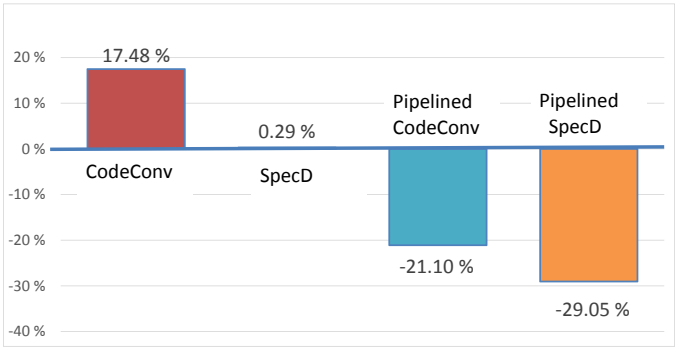


Figura 3.27: Tasa de incremento (reducción) para el retardo bajo los escenarios S1 y S2 de RBCD+RBCD$_{-5421}$ SpecD y CodeConv.

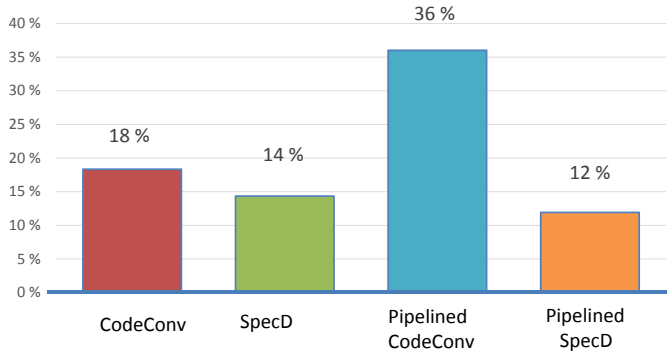


Figura 3.28: Tasa de incremento (reducción) para el área bajo los escenarios S1 y S2 de RBCD+RBCD$_{-5421}$ SpecD y CodeConv.

RBCD+RBCD$_{-5421}$ CodeConv segmentado se ha incrementado un 21 % y el área se ha reducido un 36 %. Por otro lado, el retardo de RBCD+RBCD$_{-5421}$ SpecD se ha incrementado un 29 % y el área se ha reducido un 12 %.

Para los árboles no segmentados (véase las figuras 3.29 y 3.30), la reducción del retardo obtenido bajo S2 de los árboles basados en RBCD+RBCD$_{-5421}$ SpecD (sobre un 13 %-23 %) es menor que la reducción del retardo de los árboles basados en RBCD+RBCD$_{-5421}$ CodeConv (entre 21 %-27 %). Esto se debe a que olDFA$_{Mformat}$ mediante diseño específico (SpecD) tiene más área que el olDFA$_{Mformat}$ mediante etapa de conversión (CodeConv). Por ello, el balanceo obtenido por la herramienta es más efectivo con respecto al área de los árboles

UNIVERSIDAD DE MÁLAGA

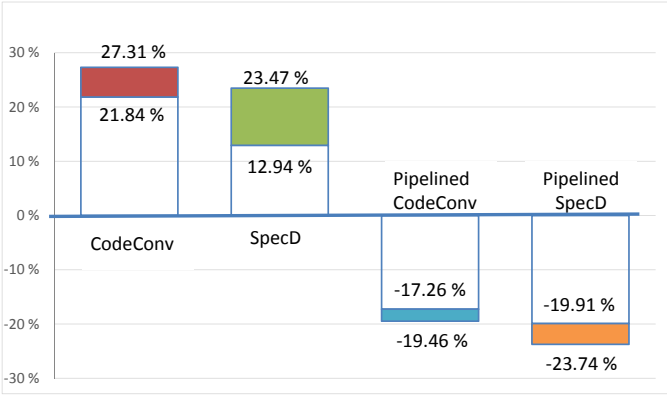


Figura 3.29: Tasa de incremento (reducción) para el retardo bajo los escenarios S1 y S2 de los árboles basados en olDFA$_{Mformat}$s.

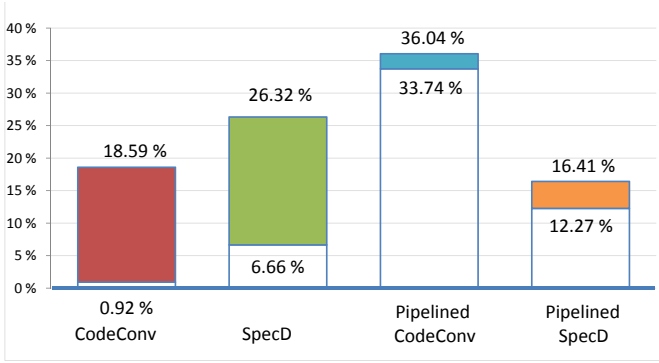


Figura 3.30: Tasa de incremento (reducción) para el área bajo los escenarios S1 y S2 de los árboles basados en olDFA$_{Mformat}$s.

mediante diseño específico (SpecD) (6 %-26 %) que al área de los árboles mediante etapa de conversión (CodeConv) (1 %-18 %).

La comparación de los resultados obtenidos para los árboles segmentados bajo S2 con los de S1 (véase las figuras 3.29 y 3.30) muestra que el área de los árboles segmentados mediante etapa de conversión (CodeConv) y diseño específico (SpecD) se reduce entre 33 %-36 % y 12 %-16 % respectivamente. Sin embargo, el retardo de los árboles segmentados mediante etapa de conversión (CodeConv) y diseño específico (SpecD) se incrementa entre 17 %-19 % y 19 %-23 % respec-

UNIVERSIDAD DE MÁLAGA

tivamente. Por ello, el balanceo obtenido bajo S2 es más efectivo con respecto al área y retardo para los árboles segmentados mediante etapa de conversión (CodeConv).

Por todo ello, el mejor escenario para las arquitecturas no segmentadas es S2 debido al hecho de que la herramienta reduce el retardo y el área comparado con S1. Sin embargo, el mejor escenario para el retardo de las arquitecturas segmentadas es S1 y para obtener el mejor área es S2.

3.3. Ejemplos de Arquitecturas Específicas para Cálculos Financieros

Hemos señalado anteriormente que en la literatura se han utilizado formatos decimales con distintos pesos para optimizar algunos algoritmos decimales. En concreto, el CORDIC decimal utiliza los formatos decimales con peso 5221, 5421 [40] y 5211 [39]. En [36] y en [38] se utilizan los códigos 4221 y 5421 para desarrollar multiplicadores decimales de alto rendimiento.

En esta sección se presenta el diseño de algunas arquitecturas específicas para realizar cálculos realizados en operaciones financieras, y donde el multiformato puede ser requerido. Más específicamente la primera arquitectura propuesta calcula diferentes aspectos financieros, como el total de un préstamo, su anualidad o su tasa de interés, así como la tasa de interés de cualquier inversión financiera [15]. Para ello se utiliza la ecuación $V_a = \sum_{j=1}^{n} C_j * (1+i)^{-j}$, donde V_a es el valor actual de una anualidad, C_j es el termino j de una anualidad e i es la tasa de interés por periodo ($(1+i)^j$ está tabulado).

Para realizar la multiplicación de la ecuación, usamos el multiplicador decimal paralelo de alto rendimiento presentado en [36], el cual utiliza la codificación RBCD_{4221}. Para la suma, usamos un modulo $\text{olDFA}_{Mformat}$, el cual tiene una entrada con el formato RBCD_{4221} y la otra entrada con el formato RBCD. La figura 3.31.a muestra los módulos de la arquitectura. Ya que el multiplicador decimal de [36] es paralelo, se serializa la salida del multiplicador (MSD primero) y se conecta al modulo $\text{olDFA}_{Mformat}$. La salida del módulo $\text{olDFA}_{Mformat}$ puede ser usada para obtener el resultado final o para ser conectada con otro modulo online si es necesario. La figura 3.31.a muestra una sincronización aproximada, donde el módulo multiplicador tiene un retardo alto (multiplicador paralelo) y el módulo $\text{olDFA}_{Mformat}$ tiene un retardo menor pero necesita más ciclos. Ambos módulos trabajan en paralelo. La ventaja de esta arquitectura es que el hardware necesario para realizar la suma es un 80 % menor que el de un diseño con un

UNIVERSIDAD DE MÁLAGA

sumador paralelo, como se muestra en la siguiente subsección.

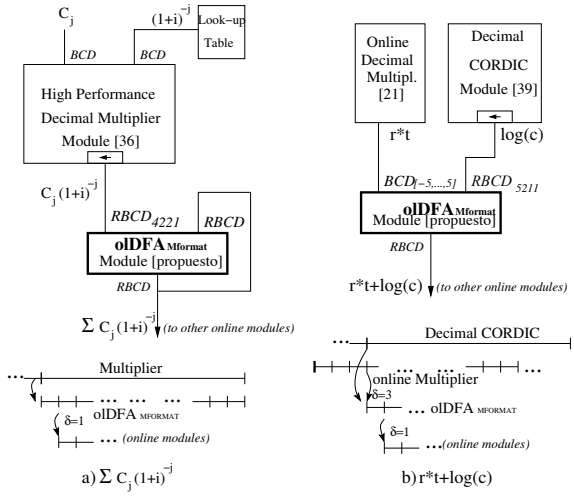


Figura 3.31: Arquitecturas para calcular funciones financieras.

La segunda arquitectura calcula el precio de un contrato futuro que está relacionado con su activo subyacente por el teorema de paridad de mercados futuros y spot (spot-futures parity), que establece que el precio futuro debe estar relacionado con el precio de contado [3]. Para ello se utiliza la ecuación $F = log(c) + r \cdot t$, siendo F una función del precio futuro, c es el precio al contado, r es el coste total de desarrollo y t es el momento de madurez. El precio de un contrato futuro está relacionado con su activo subyacente mediante el teorema de paridad de mercados futuros y spot.

Proponemos el uso de un algoritmo decimal CORDIC presentado en [39] (donde se usa la codificación RBCD_{5211}) para calcular $log(c)$ y para calcular $r \cdot t$ utilizamos el multiplicador decimal online propuesto en [21] donde se usa la codificación BCD con el rango de dígitos -5,...,5, el cual es un subrango del rango de la codificación RBCD por lo que una conversión online no es necesaria en el caso del multiplicador. El sumador $\text{olDFA}_{Mformat}$ se utiliza para sumar la salida del CORDIC decimal y el multiplicador decimal online obteniendo una salida RBCD online, como se muestra en la figura 3.31.b. En este caso, el elemento más lento es el módulo CORDIC ya que es una implementación paralela, y la salida es serializada (MSD primero) y usada como entrada para el sumador $\text{olDFA}_{Mformat}$ (véase la sincronización aproximada en la figura 3.31.b).

UNIVERSIDAD DE MÁLAGA

3.4. Conclusiones del Capítulo

En esta sección se han presentado diferentes diseños hardware para realizar la suma decimal online multioperando y multiformato. Todos los diseños se basaron en los módulos olDFA y olDFA$_{\mathrm{p}}$ definidos previamente en la sección 2.3. Además, se han presentan algunas expresiones analíticas de las arquitecturas que resultan útiles para realizar estudios previos de los sistemas a diseñar. De los resultados obtenidos, se observa que el árbol de sumadores segmentado basado en el módulo olDFA$_{\mathrm{p}}$ tiene un 86 % más de throughput con un 30 % más de área que el basado en olDFAs.

Como el uso de formatos de representación específicas permiten la optimización de algoritmos decimales, hemos presentado además, en este capítulo, dos estrategias para diseñar sumadores decimales online multioperando y multiformato. La primera estrategia se basa en un olDFA con una etapa de conversión, y la segunda se basa en el diseño específico del sumador multiformato modificando para ello la arquitectura interna del olDFA. La comparación de las estrategias ha mostrado que:

i) la primera es la mejor opción cuando cada entrada del sumador admite más de dos codificaciones de los operandos,

ii) si las entradas sólo pueden admitir operandos representados por sólo dos códigos, la segunda opción sería la mejor con respecto al retardo.

La suma decimal online multiformato se extendió al caso multioperando y se ha presentado una arquitectura general de los árboles correspondientes. En dichos árboles, el primer nivel está compuesto por sumadores olDFA$_{Mformat}$s mientras que el resto de niveles están compuestos por olDFAs. También se han analizado las arquitecturas segmentadas y no segmentadas de todos los sumadores multiformato propuestos.

También se ha presentado un detallado estudio de la principal arquitectura propuesta bajo dos diferentes escenarios de simulación. Bajo el primer escenario S1, el resultado teórico ha sido corroborado mediante los resultados experimentales, mientras que con el segundo escenario S2 dejamos a la herramienta de sintetización optimizar el diseño para que obtenga un balance entre área y retardo. Basándonos en estos resultados experimentales, podemos concluir que para las arquitecturas no segmentadas el mejor escenario es S2, mientras que para las arquitecturas segmentadas el mejor escenario es S1.

UNIVERSIDAD DE MÁLAGA

Finalmente, presentamos unas arquitecturas específicas para realizar cálculos financieros que se pueden beneficiar del uso del sumador multiformato.

En resumen, en esta sección hemos presentado una pauta para el diseño de sumadores decimales online multiformato y multioperando. Esta pauta puede servir a los diseñadores a decidir cuál es la mejor opción para sus aplicaciones.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

4 Multiplicador Decimal Online

4.1. Introducción

Hasta la fecha, se han propuesto varios diseños de multiplicadores decimales paralelos usando sistemas de codificación redundantes para reducir el retardo y el hardware [16, 35, 17]. Sin embargo, la literatura es escasa con respecto a los multiplicadores decimales online. De hecho, solo se ha propuesto un multiplicador decimal online en punto flotante [21]. Nuestro algoritmo de multiplicación trata números enteros, mientras que el propuesto en [21] trata números en punto flotante con una mantisa normalizada en el rango $10^{-1} < X < 1$.

En este capítulo presentaremos el diseño de un multiplicador decimal online junto con una evaluación de nuestro diseño y los resultados los comparamos con los del multiplicador decimal paralelo más rápido en la literatura [35]. La evaluación se lleva a cabo en términos de área y del tiempo necesario para obtener el MSD en un sistema online.

4.2. Multiplicación Decimal Online

En esta sección, se presenta una arquitectura que opera con números decimales enteros codificados en RBCD. El algoritmo de la multiplicación online que proponemos está basado en el algoritmo de multiplicación online presentado en [8]. Dicho algoritmo lo hemos adaptado para realizar la computación con n-dígitos

UNIVERSIDAD DE MÁLAGA

enteros en RBCD (números no normalizados). Debido a que usamos la representación RBCD, la función de selección propuesta en el algoritmo presentado en [8] no es necesaria debido a que la precisión de las operaciones es exacta y el resultado es representado en RBCD.

Para desarrollar el algoritmo para la multiplicación online en radix-10 necesitamos utilizar registros internos que se irán actualizando en cada ciclo y que se usarán para ir almacenando lo que se denota como residuo acumulado en cada ciclo. En concreto, se utilizarán los registros internos denotados como residuos w y v, donde v es una parte del residuo w y se usa para para obtener el dígito de salida mediante un método de truncamiento.

A continuación se pasa a explicar el algoritmo online de multiplicación para números decimales. Sean los operandos RBCD x e y, el resultado del producto p, y el número de dígitos de los operandos n. Las expresiones de los operandos y el resultado del producto en el ciclo j son las siguientes:

$$x[j] = \sum_{i=n-1}^{n-1+j+\delta} x_i 10^i, y[j] = \sum_{i=n-1}^{n-1+j+\delta} y_i 10^i$$

$$p[j] = \sum_{i=2n-1}^{2n-1-j} p_i 10^i$$

donde δ representa el retardo online (los ciclos necesarios para obtener el primer dígito del resultado). En el caso de la multiplicación, $\delta = 3$ ya que se necesita, al menos, tres dígitos de los operandos para obtener el valor correcto del MSD del resultado del producto [8].

La cota de error producida en el ciclo j es:

$$x[j] \cdot y[j] - p[j] \leq 10^j$$

El residuo producido en el ciclo j es definido como:

$$w[j] = 10^j \cdot (x[j] \cdot y[j] - p[j])$$

Los algoritmos online suelen consistir en recurrencias en valores numéricos. En concreto, para el algoritmo de multiplicación la recurrencia resultante es:

$$w[j+1] = 10 \cdot w[j] + (x[j] y_{\text{n-1-(j+3)}} + y[j+1] x_{\text{n-1-(j+3)}}) - p_{\text{2n-(j+1)}}$$

UNIVERSIDAD DE MÁLAGA

El residuo w se puede descomponer en:

$$v[j] = 10 \cdot w[j] + (x[j]y_{\text{n-1-(j+3)}} + y[j+1]x_{\text{n-1-(j+3)}})$$

$$w[j+1] = v[j] - p_{\text{2n-(j+1)}}$$

donde la salida, $p_{\text{2n-(j+1)}}$, en el ciclo j se obtiene mediante la truncación del residuo v:

$$p_{\text{2n-(j+1)}} = \widehat{v}[j]$$

El residuo estimado de v, $\widehat{v}[j]$ se obtiene truncando el valor de v y tomando solo el MSD del mismo.

RADIX-10 ONLINE MULTIPLICATION ALGORITHM

Step1: Initialize

x[-3] = y[-3] = w[-3] = 0

for j = -3;-2;-1

x[j + 1] $\leftarrow$ A(x[j]; $x_{\text{n-1-(j+3)}}$); y[j + 1] $\leftarrow$ A(y[j]; $y_{\text{n-1-(j+3)}}$)

v[j] = 10·w[j] + (x[j]$y_{\text{n-1-(j+3)}}$ + y[j + 1]$x_{\text{n-1-(j+3)}}$)

w[j + 1] $\leftarrow$ v[j]

end for

Step2: Recurrence:

for j = 0….n -1

x[j + 1] $\leftarrow$ A(x[j]; $x_{\text{n-1-(j+3)}}$); y[j + 1] $\leftarrow$ A(y[j]; $y_{\text{n-1-(j+3)}}$)

v[j] = 10·w[j] + (x[j]$y_{\text{n-1-(j+3)}}$ + y[j + 1]$x_{\text{n-1-(j+3)}}$)

$p_{\text{2n-(j+1)}} = \widehat{v}[j]$

w[j + 1] = v[j] - $p_{\text{2n-(j+1)}}$

end for

Figura 4.1: Algoritmo de multiplicación decimal online.

UNIVERSIDAD DE MÁLAGA

La figura 4.1 muestra con más detalle el algoritmo de la multiplicación online propuesto en este capítulo. Como puede verse, el algoritmo consiste en dos partes. La primera es la inicialización del residuo w (con los ciclos $j = -3, \cdots, -1$), y la segunda parte consiste en el desarrollo de la recurrencia en el residuo $w[j]$. La función A es una función de inserción que desplaza 4 bits a la izquierda el vector para insertar el dígito de entrada del ciclo correspondiente en la posición menos significativa.

La fase de recurrencia del algoritmo obtiene el dígito del producto empezando por el más significativo y acumula el residuo generado debido a la falta de datos característico de la aritmética online, y así compensar dicho residuo en los futuros ciclos.

A la hora de implementar el algoritmo de multiplicación decimal online, exploramos dos arquitecturas diferentes. La primera de ellas consiste en una arquitectura de multiplicación decimal online sin especulación descrita en la sección 4.2.1, y la otra arquitectura consiste en una multiplicación decimal online con especulación con el objetivo de reducir el alto coste de computación en cada ciclo descrito en la sección 4.2.2.

Para obtener $v[j]$ se necesita multiplicar dos vectores ($x[j]$ e $y[j+1]$) por un dígito ($y_{\text{n-1-(j+3)}}$ y $x_{\text{n-1-(j+3)}}$, respectivamente), es decir, se necesita realizar dos multiplicaciones vector por dígito. Estas multiplicaciones solucionan el error producido por los dígitos de entrada en el ciclo j, ($x_{\text{n-1-(j+3)}}$ e $y_{\text{n-1-(j+3)}}$) , debido a que dichos dígitos no son conocidos en el ciclo anterior $j-1$.

4.2.1. Multiplicación Decimal Online sin Especulación

Para la implementación del algoritmo presentado en la figura 4.1 proponemos una multiplicación decimal online sin especulación usando la arquitectura mostrada en la figura 4.2. Los vectores de entrada $x[n]$ y $y[n]$ son inicializados con n dígitos a cero. En cada ciclo j, los vectores son actualizados incluyendo los dígitos de entrada en las posiciones $n-1-(j+3)$. Debemos recalcar que los dígitos de entrada, $y_{\text{n-1-(j+3)}}$ y $x_{\text{n-1-(j+3)}}$ multiplican a los vectores $x[j]$ e $y[j+1]$. Es por ello que dicha arquitectura contiene dos módulos vector por dígito que obtienen un resultado en Carry-Save (cs y ps) cada uno. La arquitectura del módulo vector por dígito se decribe a continuación.

UNIVERSIDAD DE MÁLAGA

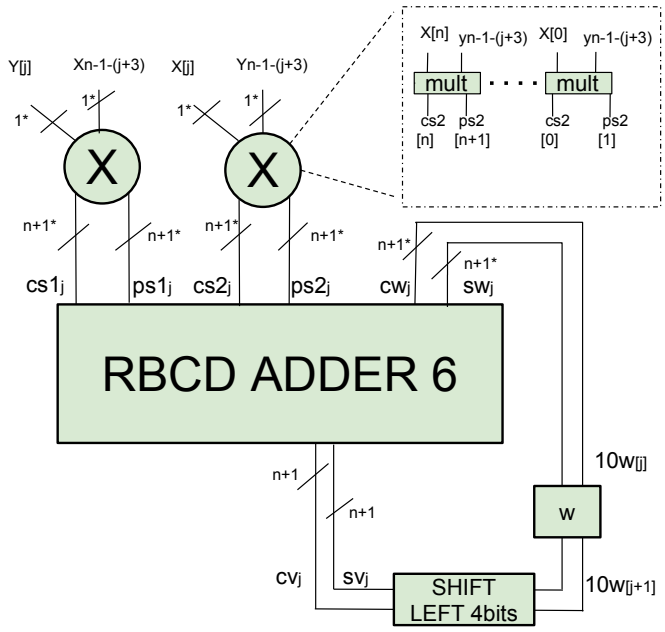


Figura 4.2: Arquitectura general del módulo de multiplicación vector por dígito sin especulación.

Vector por Dígito sin Especulación

La multiplicación vector por dígito sin especulación realiza la multiplicación en paralelo de cada dígito de los vectores $y[j+1]$ y $x[j]$, por el dígito de entrada en cada ciclo $x_{n-1-(j+3)}$ o $y_{n-1-(j+3)}$. Cada dígito del vector será la entrada del módulo multiplicador paralelo, siendo éste un módulo de multiplicación de dígito por dígito obteniendo un resultado en Carry-Save, *cs* y *ps* como se muestra en la parte superior derecha de la figura 4.2.

Para obtener el resultado de $v[j]$, debemos sumar los cuatro dígitos RBCD producidos por los módulos vector por dígito y el residuo w representado en Carry-Save. A continuación se describe la arquitectura implementada para la suma paralela de dichos dígitos.

UNIVERSIDAD DE MÁLAGA

Sumador de 6 dígitos RBCD

Para realizar la suma paralela de los seis dígitos ($cs1$, $ps1$, $cs2$, $ps2$, cw y sw) se ha implementado la arquitectura mostrada en la figura 4.3.

El modulo **ext** realiza una extensión de signo en un bit más.

Por otro lado, los módulos adder6, adder5 y adder4 son Full Adders de 6 bits, 5 bits y 4 bits respectivamente.

La salida del último sumador de 6 bits es transformada a dos dígitos RBCD (cv y sv) ya que el dígito de 6 bits puede representar números dentro del rango {-32,...,31}. La suma de los seis dígitos está dentro de dicho rango ya que los valores de los dígitos sw, $sp1$ y $sp2$ están dentro del rango {-6,..,6} , los valores de los dígitos $cs1$ y $cs2$ están dentro del rango {-5,..,5}, además del valor de cw que está dentro del rango {-3,..,3}. La suma de todos los rangos es de {-31,..,31}.

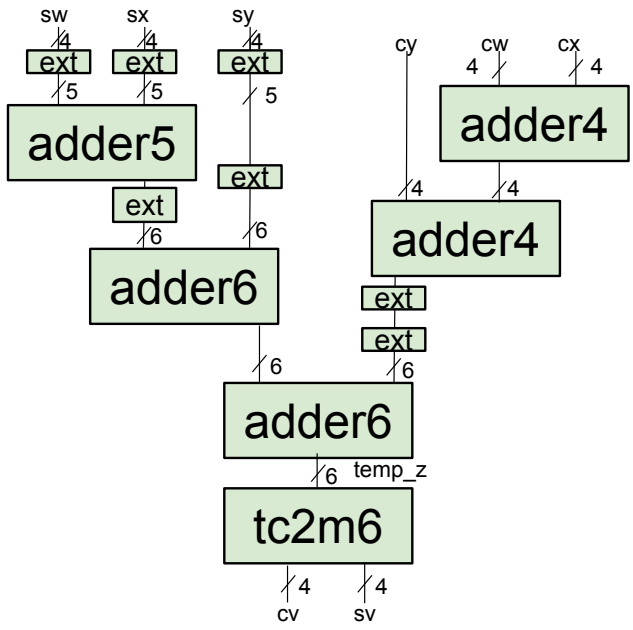


Figura 4.3: Arquitectura general del módulo de suma de 6 dígitos RBCD.

UNIVERSIDAD DE MÁLAGA

4.2.2. Multiplicación Decimal Online con Especulación

El diagrama de bloque del algoritmo de la multiplicación decimal online con especulación se muestra en la figura 4.4. Los vectores de entrada $x[n]$ e $y[n]$ son inicializados con n dígitos a cero. En cada ciclo j, los vectores son actualizados incluyendo los dígitos de entrada en las posiciones $n-1-(j+3)$. Debemos recalcar que los dígitos de entrada, $y_{\text{n-1-(j+3)}}$ y $x_{\text{n-1-(j+3)}}$ multiplican a los vectores $x[j]$ e $y[j+1]$, respectivamente. Los resultados de las multiplicaciones vector por dígito se suman usando $n+1$ sumadores RBCD paralelos [16] (implementados usando las ecuaciones corregidas mostradas en la ecuación 2.3). El resultado de esta suma es añadido con el residuo desplazado del ciclo anterior (cuya expresión es $10 \cdot w[j]$). Para ello, usamos $n+3$ sumadores RBCD. Cuando desplazamos 4 bits el estado interno $v[j]$, obtenemos el residuo para el siguiente ciclo $w[j+1]$ multiplicado por 10. Tenemos que mencionar que al desplazar a la izquierda el vector v, se descarta el dígito más significativo, es decir, la resta $w[j+1] \leftarrow v[j] - p_{2\text{n-(j+1)}}$ y la multiplicación del residuo w por 10 son obtenidas desplazando el registro interno $v[j]$ 4 bits a la izquierda.

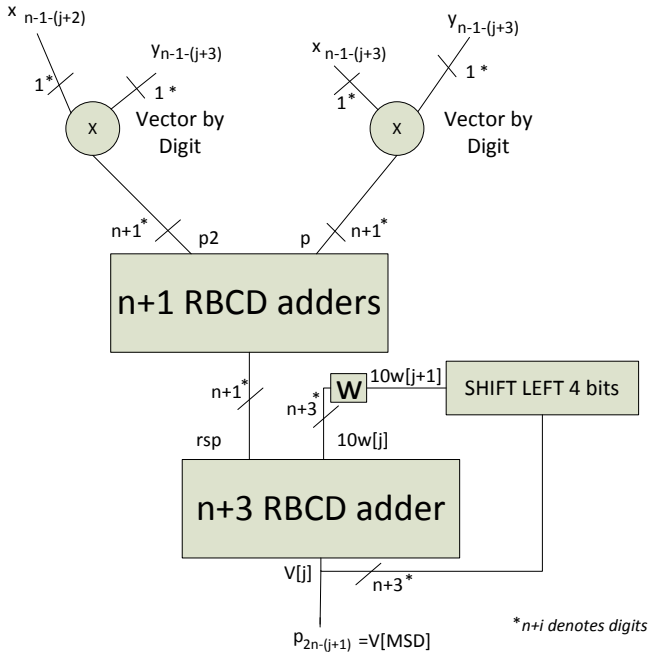


Figura 4.4: Arquitectura de la multiplicación decimal online con especulación.

UNIVERSIDAD DE MÁLAGA

De los 4 bits más significativos de $v[j]$ obtenemos el dígito $p_{2n-(j+1)}$ del resultado de la multiplicación.

La multiplicación vector por dígito con especulación que acabamos de presentar obtiene en cada ciclo los vectores resultados para cada valor absoluto del dígito de entrada posible {0, 1, 2, 3, 4, 5, 6, 7}. Por ello el cálculo en cada ciclo es el nuevo dígito del vector multiplicado por cada uno de los posibles valores del dígito de entrada y, una vez obtenido, es sumado al vector resultado de cada valor. El vector resultado se obtiene de un multiplexor 8 a 1 utilizándose el dígito de entrada como señal de control.

Vector por Dígito con Especulación

Con el objetivo de reducir el alto coste de computación de la multiplicación de todos los dígitos del vector por el dígito en cada ciclo, se ha diseñado un esquema de especulación. Cada dígito de los vectores $y[j+1]$ y $x[j]$ son multiplicados por los múltiplos de los dígitos RBCD en valor absoluto {1X, 2X, 3X, 4X, 5X, 6X, 7X} y el resultado es almacenado en vectores Carry-Save {p_1[i], c_1[i], s_2[i], c_2[i], p_3[j], c_3[i], p_4[i], c_4[i], p_5[i], c_5[i], p_6[i], c_6[i], p_7[i], c_7[i]} codificados en RBCD, donde i va desde 0 a n-1. Las salidas producidas por el multiplicador son las entradas de dos multiplexores 8-1 de n-dígitos utilizándose el valor absoluto del dígito $x_{n-1-(j+3)}$ o $y_{n-1-j+3}$ como control para obtener el resultado del vector en Carry-Save.

La arquitectura del módulo de la multiplicación vector por dígito se muestra en la figura 4.5. Este módulo está compuesto por un módulo multiplicador, un multiplexor 8-1, un registro y tres multiplexores 2-1.

A continuación vamos a explicar cómo funciona el módulo vector por dígito suponiendo que estamos en el ciclo j y estamos calculando el producto de $x[j]$ por $y_{n-1-(j+3)}$. El módulo recibe como dígitos de entrada $x_{n-1-(j+2)}$ e $y_{n-1-(j+3)}$. Este último dígito se almacena en un registro para ser usado en el siguiente ciclo. El signo del dígito $x_{n-1-(j+2)}$ se utiliza para calcular el valor absoluto de $x_{n-1-(j+2)}$ (escogiendo entre el dígito y su Complemento a Dos). El módulo multiplicador recibe el signo y el valor absoluto de $x_{n-1-(j+2)}$ y realiza la multiplicación especulativa. Las salidas del módulo del multiplicador son vectores con signo. Para seleccionar el registro multiplicado adecuado, usamos el valor absoluto de $y_{n-1-(j+2)}$. Y por último, usamos de nuevo los signos de $x_{n-1-(j+2)}$ e $y_{n-1-(j+2)}$ para seleccionar entre el resultado positivo o negativo del producto del vector por dígito ($p2$ en la figura 4.4).

UNIVERSIDAD DE MÁLAGA

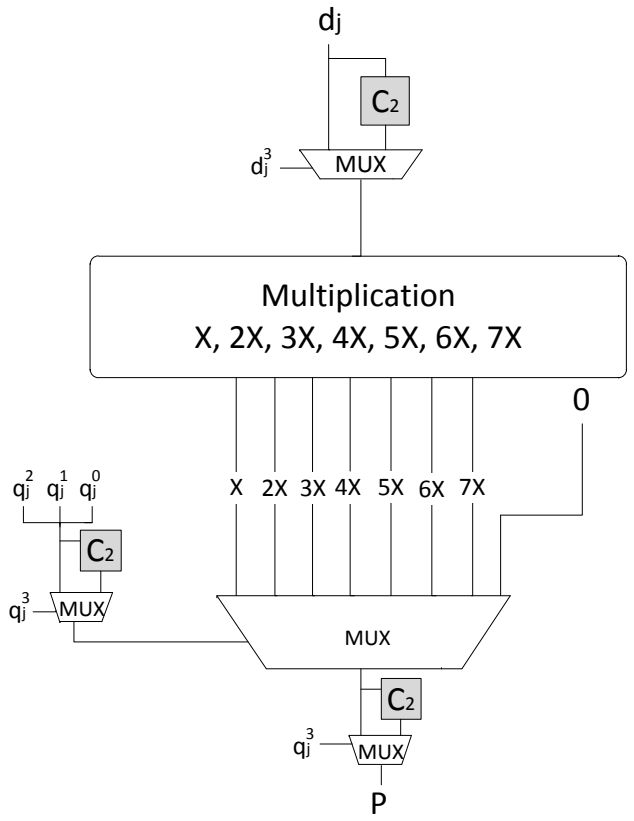


Figura 4.5: Arquitectura general del módulo de multiplicación vector por dígito con especulación.

Módulo Multiplicador

La arquitectura del módulo multiplicador se muestra en la figura 4.6. Las Tablas 4.1 y 4.2 describen la tabla de verdad de dicho módulo, el cual consiste en la multiplicación del dígito RBCD (x_k) por un dígito en el rango {2,..,7}, obteniendo un producto (p_k) con un acarreo (t_k) para cada uno. Así, para una entrada, el módulo produce 6 productos. Por lo tanto, tenemos que sumar los productos del módulo multiplicador p_k con el acarreo obtenido en el ciclo previo t_k. Por todo ello, necesitamos implementar olDFAs para obtener el resultado de la suma para cada producto como se muestra en la figura 4.6.c. Hemos conseguido reducir área

UNIVERSIDAD DE MÁLAGA

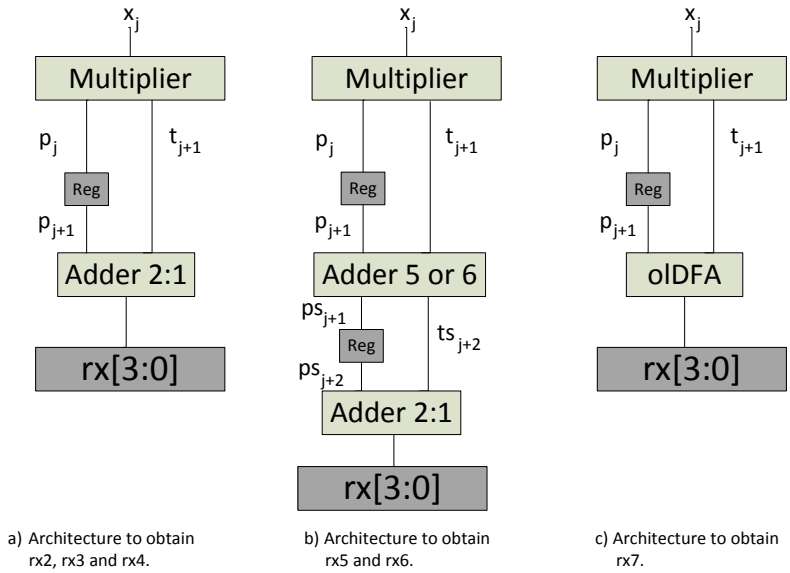


Figura 4.6: Arquitectura del módulo multiplicador.

Dígito	Mult. 2			Mult. 3			Mult. 4		
	t_{k+1}	p_k	x_k	t_{k+1}	p_k	x_k	t_{k+1}	p_k	x_k
7	1	0100	4	10	0001	1	11	1110	-2
6	1	0010	2	10	1110	-2	10	0100	4
5	1	0000	0	01	0101	5	10	0000	0
4	1	1110	-2	01	0010	2	10	1100	-4
3	1	1100	-4	01	1111	-1	01	0010	2
2	0	0100	4	01	1100	-4	01	1110	-2
1	0	0010	2	00	0011	3	00	0100	4
0	0	0000	0	00	0000	0	00	0000	0

Tabla 4.1: Tabla de verdad del módulo multiplicador para la multiplicación del dígito x_k con los dígitos 2, 3 y 4

y retardo reemplazando los olDFAs por sumadores específicos para todos los casos excepto para el multiplicador por 7 debido al rango de los valores de la salida. La figura 4.6.a muestra los casos para la suma de las salidas multiplicadas por 2, 3 y 4, donde la suma de los rangos de los productos y acarreos no exceden el rango de la codificación RBCD. Por ello, en estos casos la suma necesita un registro (el cual almacena el producto previo (p_{k+1}) y el otro operando es tomado de la salida del multiplicador (t_{k+1}), reduciendo el área y el retardo considerablemente.

UNIVERSIDAD DE MÁLAGA

Dígito	Mult. 5			Mult. 6			Mult. 7		
	t_{k+1}	p_k	x_k	t_{k+1}	p_k	x_k	t_{k+1}	p_k	x_k
7	11	0101	5	100	0010	2	101	1111	-1
6	11	0000	0	100	1100	-4	100	0010	2
5	10	0101	5	011	0000	0	011	0101	5
4	10	0000	0	010	0100	4	011	1110	-2
3	01	0101	5	010	1110	-2	010	0001	1
2	01	0000	0	001	0010	2	001	0100	4
1	00	0101	5	001	1100	-4	001	1101	-3
0	00	0000	0	000	0000	0	000	0000	0

Tabla 4.2: Tabla de verdad del módulo multiplicador para la multiplicación del dígito x_k con los dígitos 5, 6 y 7

Para los casos de los dígitos multiplicados por 5 y 6, el rango de la suma de p_k más el acarreo exceden el rango de la codificación RBCD (el rango de p_k es {-4,...,5} y el rango de los acarreos es {0,...,4}). Usando un sumador específico, la suma (ps_{k+1}) está dentro del rango {-6,...,6} y el acarreo (ts_{k+2}) está dentro del rango {-1,0,1}. A continuación se suman el dígito de acarreo ts_{k+2} y la suma previamente almacenada ps_k usando un sumador binario, obteniendo un retardo online de 2. Las ecuaciones que implementa el módulo Adder5 son las presentadas en la ecuación 4.1. Este sumador se usa para reducir el área y retardo, ya que la suma del múltiplo de 5 {0,5} con el acarreo producido {0,1,2,3} es más simple que la suma de dos dígitos RBCD utilizando un olDFA.

$$\begin{aligned}
&ps_k0 = x_k0 \oplus y_k0 \\
&ps_k1 = x_k1 \cdot \overline{y_k0} + x_k0 \cdot (y_k2 \oplus (\overline{y_k1 \oplus y_k0}) \\
&ps_k2 = \overline{x_k0} \cdot y_k2 + x_k0 \cdot (\overline{y_k2} \cdot (y_k1 + y_k0) + y_k1 \cdot y_k0) \\
&ts_{k+1}3 = x_k0 \oplus y_k3 \\
&ts_{k+1}0 = x_k3 \cdot y_k3 + x_k2 \cdot \overline{y_k3} \\
&ts_{k+1}1 = t_k2 = t_k3 = x_k3 \cdot y_k3
\end{aligned} \tag{4.1}$$

La ecuación 4.2 muestra las funciones implementadas del sumador personalizado para las salidas multiplicadas por 6.

UNIVERSIDAD DE MÁLAGA

$$
\begin{aligned}
ps_k^0 &= y_k^0 \\
ps_k^1 &= y_k^3 \cdot ((x_k^3 + x_k^1) \oplus y_k^1) + \overline{y_k^3} \cdot \overline{x_k^3 \oplus x_k^2} \cdot (x_k^1 \oplus y_k^1) \\
&\quad + \overline{x_k^3} \cdot x_k^2 \cdot \overline{y_k^3} \cdot \overline{y_k^1} \\
ps_k^2 &= x_k^3 \cdot x_k^2 \cdot (\overline{y_k^2} \oplus y_k^1 + \overline{x_k^1} \cdot \overline{y_k^2}) + \overline{x_k^2} \cdot \overline{x_k^1} \cdot y_k^2 \\
&\quad + \overline{x_k^2} \cdot x_k^1 \cdot (y_k^2 \oplus y_k^1) + \overline{x_k^3} \cdot x_k^2 \cdot \overline{y_k^3} \cdot (y_k^1 + y_k^2) \\
ps_k^3 &= x_k^3 \cdot \overline{x_k^1} \cdot \overline{y_k^2} + x_k^3 \cdot x_k^1 \cdot (y_k^3 + \overline{y_k^2} \cdot \overline{y_k^1}) \\
&\quad + \overline{x_k^3} \cdot x_k^1 \cdot y_k^3 \cdot \overline{y_k^1} + \overline{x_k^3} \cdot \overline{x_k^1} \cdot (x_k^2 \oplus y_k^3) \\
ts_{k+1}^3 &= x_k^3 \cdot \overline{x_k^1} \cdot y_k^3 \\
ts_{k+1}^0 &= ts_{k+1}^1 = ts_{k+1}^2 = ts_{k+1}^3
\end{aligned}
\tag{4.2}
$$

Una vez que se obtienen los múltiplos sin signo, el signo del dígito x_k se usa para seleccionar entre el vector positivo o negativo del múltiplo.

4.3. Resultados Experimentales

Todas las arquitecturas presentadas en este apartado se han modelado y verificado a nivel RTL con Verilog. También, se han sintetizado usando Synopsis Design Compiler y la librería de celdas TSMC's tcbn65gplus 65 nm CMOS. Todo el entorno y parámetros de proceso se fijaron a las condiciones normales o típicas.

En esta sección, primero comparamos los resultados obtenidos del multiplicador decimal online propuesto sin especulación (véase la sección 4.2.1) y con especulación (véase la sección 4.2.2). Y a continuación compararemos el multiplicador propuesto con el multiplicador decimal paralelo más rápido en la literatura [35].

4.3.1. Especulación vs No Especulación

La Tabla 4.3 muestra los resultados obtenidos para las dos arquitecturas propuestas. Como puede verse, la arquitectura de la multiplicación decimal online sin especulación obtiene una reducción del 58 % en área en comparación con la arquitectura con especulación mientras que en términos de retardo las dos

UNIVERSIDAD DE MÁLAGA

Arquitectura	Retardo (ns)	Área (μm^2)	Retardo Total (ns)	Retardo MSD (ns)
Sin especulación	1	23861	20.44	4.44
Con especulación	0.98	57233	20.04	4.36

Tabla 4.3: Resultados del multiplicador decimal online sin especulación y con especulación

Arquitectura	Retardo (ns)	Área (μm^2)	Retardo Total (ns)	Retardo MSD (ns)
Nuestro multiplicador online	1	23861	20.44	4.44
[35]	1.65	61049	5.59	5.59

Tabla 4.4: Resultados del multiplicador online y el multiplicador paralelo

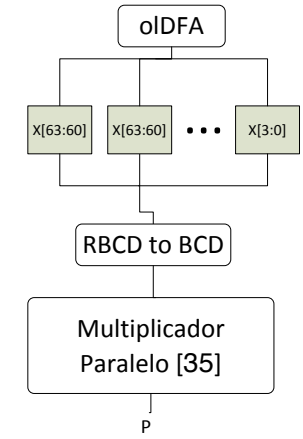
arquitecturas son similares. Hay que resaltar que aunque la arquitectura con especulación es más rápida, el gran incremento del área al utilizar éste método junto con el moderado incremento de velocidad hace que lo debamos descartar para una comparación con el multiplicador decimal paralelo.

4.3.2. Comparación con Multiplicador Paralelo

En los sistemas online, el retardo para obtener el dígito más significativo del resultado es un parámetro importante ya que impone el intervalo de iniciación, que es el tiempo necesario para solapar el cálculo actual con otra operación online. Como una comparación directa de un multiplicador online con un multiplicador paralelo no tiene sentido, para llevar a cabo una comparación justa, hemos realizado la siguiente comparación. Hemos creado dos sistemas online: uno tiene el multiplicador online que hemos propuesto y el otro tiene en su lugar el multiplicador paralelo presentado en [35]. La razón de esta elección es porque este multiplicador paralelo usa aritmética decimal con codificación BCD redundante. Hemos implementado ambas arquitecturas para un sistema decimal online de $n = 16$ dígitos que realiza la operación $(x + y) \cdot z$. La primera operación $(x + y)$ la realiza un olDFA en ambos sistemas online. La segunda operación (multiplicación por z) en uno de los sistemas es realizada por nuestro multiplicador online sin especulación y en el otro sistema es realizada por el multiplicador paralelo presentado en [35]. Ambos sistemas se muestran en la figura 4.7.

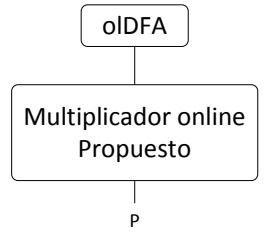
Como en la figura 4.7.a se usa un multiplicador paralelo, es necesario realizar una paralelización del resultado $(x+y)$ (el cual se obtiene de un sistema online, en este caso un sumador). Para esta paralelización son necesarios unos registros para

UNIVERSIDAD DE MÁLAGA



Module	Delay (ns)	Cycles	Total Delay (ns)	Area (μm²)
olDFA	0.22	17	3.74	395
RBCD to BCD	0.08	1	0.08	139
Mult.Para.[35]	1.65	1	1.65	58213
BCD to RBCD	0.12	1	0.12	2302
			5.59	61049

a. Arquitectura con Multiplicador paralelo



Module	Delay (ns)	Cycles	Total Delay (ns)	Area (μm²)
olDFA	0.22	2	0.44	395
Mult. online	1	4	4	23861
			4.44	24256

b. Arquitectura con multiplicador Propuesto sin especulación

Figura 4.7: Arquitecturas para comparar un multiplicador paralelo con el multiplicador online propuesto.

tener el resultado paralelo de la suma. Estos registros aparecen en la figura como x[63:60], x[59:56] ... x[3:0]. El contenido de cada uno de estos registros se obtiene uno por cada ciclo de reloj (en total 17 ciclos debido al retardo online del olDFA $\delta = 1$). Una vez se consigue la paralelización, se realiza una conversión de RBCD a BCD y finalmente se usa el multiplicador paralelo de [35]. Este sistema da un tiempo total de computación para obtener el dígito más significativo (MSD) de 5.59 ns. La tabla que figura en la figura 4.7.a muestra detalladamente el retardo

UNIVERSIDAD DE MÁLAGA

de los diferentes elementos de la arquitectura.

Por el otro lado, la figura 4.7.b muestra el sistema que usa el multiplicador online que proponemos. En este caso la salida del olDFA se conecta directamente al multiplicador online, logrando un retardo total de 4.36 ns para obtener el MSD (véase la Tabla 4.4). Esto significa que nuestro sistema tiene un incremento de velocidad del 22.5 % y una reducción de área del 60 %. Como conclusión, para los sistemas online se obtiene un mejor tiempo si se usa nuestro multiplicador decimal online en vez de un multiplicador decimal paralelo.

4.4. Conclusiones del Capítulo

Los multiplicadores decimales online propuestos en este apartado han sido diseñados para ser usados en sistemas online. Proponemos un algoritmo de multiplicación decimal online usando la codificación RBCD y dos arquitecturas comparando dos formas distintas de implementación del algoritmo. La arquitectura que implementa la multiplicación decimal online sin especulación es la arquitectura más óptima ya que tiene la mejor relación entre coste computacional y coste hardware.

Hemos diseñado un multiplicador decimal online RTL de 16x16 dígitos, y se ha insertado en un sistema online. Por motivos de comparación, también hemos implementado el mismo sistema pero sustituyendo nuestro multiplicador online por un multiplicador decimal paralelo rápido, obteniendo un incremento en velocidad del 22.5 % a la hora de obtener el MSD y a su vez reduciendo el área un 60 %.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

5 División Decimal Online

5.1. Introducción

La división es la más compleja de las cuatro operaciones aritméticas básicas y más aún en la computación aritmética decimal. La clase de recurrencia de dígitos de los algoritmos de hardware de división generalmente imita el esquema de división de papel y lápiz convencional como se enseña en la escuela primaria [28]. Sin embargo existen algoritmos de división multiplicativos que producen progresivamente una aproximación del cociente, donde el número de iteraciones es logarítmicamente proporcional al tamaño del cociente [28]. Durante las últimas décadas, varios diseños de división de recurrencia de dígitos y clases multiplicativas han sido implementados en hardware [22, 42, 20, 37].

Se han llevado a cabo innovaciones en el diseño de los esquemas de división decimal para implementar en procesadores binarios, como se puede ver en [42, 27, 20, 37] . La selección de dígitos de cociente (QDS) para la divisón decimal es la principal diferencia entre los aportes citados. Se han utilizado dos técnicas para la simplificación decimal de QDS. La primera es usar un conjunto de dígitos con signo, como [-5,5] que se presenta en [37], para la representación del cociente. Esto lleva a menos comparaciones con múltiplos del divisor. La otra es introducir más reduncia en el conjunto de dígitos. Esto permite una simplificación adicional en la implementación de la función de selección, como se expone en [20] con un rango de dígitos de [-7,7] y en [27] con un rango de [-9,9].

Nuestra propuesta en esta sección se basa en el trabajo presentado en [20] por los siguientes motivos:

UNIVERSIDAD DE MÁLAGA

- Rango de dígitos [-7,7] que coincide con el rango de dígitos de la codificación RBCD utilizada durante la tesis y que se adapta a las necesidades de la aritmética online.
- Implementación solapada de QDS y cálculo del resto parcial (PRC).
- Baja latencia de los restos parciales usando sumadores redundantes.
- Simplificación de QDS mediante la representación binaria/decimal mixta de los dígitos más significativas de los restos parciales que imponen redundancia de hardware.

5.2. División Decimal Online

En esta sección se presenta el algoritmo para realizar una división decimal online utilizando el código RBCD basado en el algoritmo propuesto en [20]. Como describimos anteriormente, la aritmética online funciona en serie, por eso, se calcula sin todos los dígitos de las entradas d (dividendo) y x (divisor). Esto lo compensamos con el módulo de corrección descrito en la sección 5.2.2.

El algoritmo propuesto se describe en la Figura 5.1. Dicho algoritmo se basa en la de separar el valor del cociente q en dos variables q_H y q_L, donde el valor de q_H define si el valor de q es mayor que 5 (representado mediante 1) o si el valor de q es menor que -5 (representado mediante -1); en caso contrario se presentará mediante 0, y el valor de $q_L \in \{-2, -1, 0, 1, 2\}$.

Debido a que no se dispone de todos los datos, el algoritmo va generando un error que es compensado mediante el módulo de corrección definido en la sección 5.2.2 multiplicando el cociente acumulado por el dígito de entrada d_j para el ciclo j. El algoritmo calcula el residuo en cada ciclo para ser usado en los futuros ciclos mediante la resta al residuo acumulado del producto del vector del divisor por el dígito del cociente obtenido en el ciclo j.

El algoritmo online propuesto proporciona un modo de precisión variable usando truncamiento. La Figura 5.2 muestra la arquitectura de la divisón decimal online. La arquitectura de dicho algoritmo está compuesto de los siguientes módulos:

- Constantes de Selección para q_{Hj+1} y q_{Lj+1} (descritas en la sección 5.2.1).
- Los módulos suma de residuo (6 dígitos RBCD) (véase la sección 5.2.3).

UNIVERSIDAD DE MÁLAGA

RADIX-10 ONLINE DIVISION ALGORITHM

Step1: Initialize

$x[-3] = y[-3] = w[-3] = 0$

for $j = -3;-2;-1$

$x[j+1] \leftarrow A(x[j]; x_{n-1-(j+3)})$; $d[j+1] \leftarrow A(y[j]; y_{n-1-(j+3)})$

$w[j+1] \leftarrow x_j/100$, *i=0, …, δ-1*

end for

Step2: Recurrence:

for $j = 0....n-1$

$x[j+1] \leftarrow A(x[j]; x_{n-1-(j+3)})$; $d[j+1] \leftarrow A(d[j]; d_{n-1-(j+3)})$

$w[j] = w[j] + x_{n-1-(j+3)} \cdot 10^{-(\delta+2)}$

$correctv = Q_H[j-1] \cdot 5 \cdot d_{n-1-(j+3)}$

$q_{Hj+1} = SEL_H(10\widehat{w[j]}, D\widehat{[j+1]})$

$v[j] = w[j] - 5D[j] \cdot q_{Hj+1} - correctv \cdot 10^{-(\delta-1)}$

$correctw = Q_L[j-1] \cdot d_{n-1-(j+3)}$

$q_{Lj+1} = SEL_L(\widehat{v[j]}, D\widehat{[j+1]})$

$w[j+1] = 10 \cdot v[j] - D[j] \cdot q_{Lj+1} - correctw \cdot 10^{-(\delta-1)}$

$q_{2n-(j+1)} = q_{Lj+1} + 5 \cdot q_{Hj+1}$

end for

Figura 5.1: Algoritmo división decimal online.

- Los módulos de corrección que contiene una multiplicación vector por digito para obtener *correctv* y *correctw* (véase en la sección 5.2.2).
- Funciones de selección para q_{Hj+1} y q_{Lj+1} (véase en la sección 5.2.1).

UNIVERSIDAD DE MÁLAGA

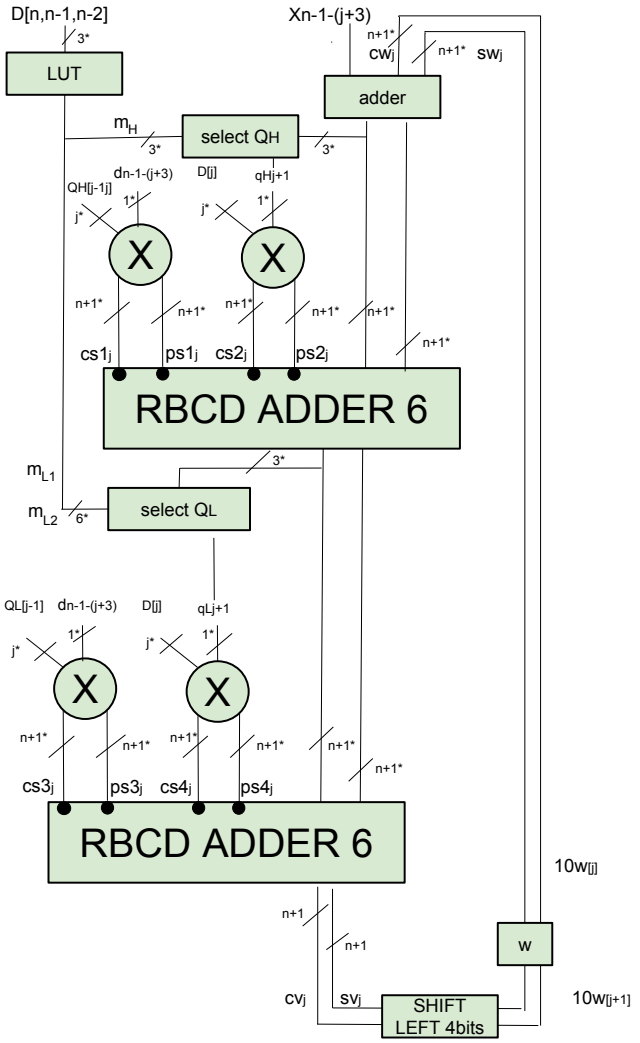


Figura 5.2: Arquitectura de la división decimal online.

5.2.1. Constantes de Selección para q_h y q_l

Las funciones de selección usan los valores truncados de $\widehat{10w}$ y $\widehat{v}$, con t números de dígitos fraccionarios. El error debido a la estimación está delimitado en $1{,}12 \cdot 10^{-t}$. Por ello, el intervalo $[L_k, U_k]$ de la constante de selección m_k es:

UNIVERSIDAD DE MÁLAGA

$$L_k(d_{i+1}) \leq U_{k-1}(d_i) - 1{,}12 \cdot 10^{-t}, \tag{5.1}$$

El intervalo de selección es obtenido con el rango de $10w[j]$, $[-10\rho d, 10\rho d]$, que por el rango de dígitos seleccionado es $[-(70/9)d, (70/9)d]$ y con el rango de $v[j]$ es obtenido con $|v[j] - q_L d| \leq \rho d$, que resulta $|v[j]| \leq (25/9)d$ con el rango de dígitos seleccionado.

Para obtener el límite superior del intervalo de selección para q_H cuando es igual a k con $k \in \{-1, 0, 1\}$ reemplazamos $10w[j]$ por el límite superior, U_k, y $v[j]$ por el valor máximo del mismo, $(25/9)$, en la ecuación:

$$v[j] = 10w[j] - 5q_{Hj+1}d$$

resultando:

$$U_k = (5k + 25/9)d$$

Lo mismo ocurre con el límite inferior del intervalo de selección para q_H, reemplazamos $10w[j]$ por el límite menor, L_k y, siendo $v[j] \geq -(25/9)$, reemplazamos $v[j]$ por el valor mínimo del mismo, $-(25/9)$.

$$L_k = (5k - 25/9)d$$

Para obtener el intervalo de selección para q_L cuando es igual que k con $k \in \{-2, -1, 0, 1, 2\}$ reemplazamos $v[j]$ por el límite superior, U_k, y $w[j]$, siendo $w[j] \leq (7/9)$, por el valor máximo del mismo, $(7/9)$, en la ecuación:

$$w[j] = v[j] - q_{Lj+1}d$$

Obtenemos:

$$U_k = (k + 7/9)d$$

Y con el valor mínimo de $w[j] \geq -(7/9)$ y el limite inferior L_k, obtenemos:

$$L_k = (k - 7/9)d$$

UNIVERSIDAD DE MÁLAGA

$[d, d_{i+1}]$	q_H		q_L			
	m_{H1}	m_{H0}	m_{L2}	m_{L1}	m_{L0}	m_{L-1}
0.100,0.106	26	-26	16	4	-4	-16
0.106,0.120	28	-28	20	8	-8	-20
0.12,0.13	32	-32	20	8	-8	-20
0.13,0.14	34	-34	20	8	-8	-20
0.14,0.15	36	-36	20	8	-8	-20
0.15,0.17	40	-40	24	8	-8	-24
0.17,0.20	46	-46	28	8	-8	-28
0.20,0.23	52	-52	32	8	-8	-32
0.23,0.25	58	-58	36	8	-8	-36
0.25,0.30	68	-68	40	8	-8	-40
0.30,0.35	80	-80	48	16	-16	-48
0.35,0.42	96	-96	56	16	-16	-56
0.42,0.50	114	-114	68	16	-16	-68
0.50,0.60	136	-136	84	16	-16	-84
060,0.70	164	-164	104	32	-32	-104
0.70,0.84	188	-188	112	32	-32	-112
0.84,1.00	224	-224	124	32	-32	-124

Tabla 5.1: Constantes de selección

Con estos intervalos determinamos las constantes de selección dividiendo el intervalo de d en subintervalos con dimensión $\Delta d = 10^{-\delta}$.

Usamos las constantes de selección de 5.1 con $t = 2$ y a continuación obtenemos $m_{-k+1} = -m_k$ para q_H y q_L, y calcula $m_H(i) = m_{H1}$ y $m_L(i) = m_{L2}, m_{L1}$ para cada $\widehat{d}$. Usando este procedimiento obtenemos la Tabla 5.1.

5.2.2. Módulo de Corrección, Vector por Dígito

Como se ha descrito anteriormente, estamos calculando un resultado sin tener todos los dígitos de las entradas d y x; por ello, necesitamos compensar dicha falta de información. La información de entrada se va completando en cada ciclo por lo que necesitamos multiplicar los valores calculados anteriormente (vector) por el nuevo dato de entrada. Es por ello que las multiplicaciones de los dígitos $d_{n-1-(j-3)}$ y q_{hj+1} por los vectores Q_H[j] y $D[j]$ respectivamente corrigen el error que se produce debido a que la entrada d se vaya completando cada ciclo.

$$correctv = (Q_H[j] * 5 * d_{n-1-(j-3)} * + D[j] * q_{hj+1})10^{-\delta+1}$$

El mismo problema ocurre con las multiplicaciones de los dígitos $d_{n-1-(j-3)}$ y q_{lj+1} por los vectores Q_L[j] y $D[j]$, por lo que corregimos el error producido

UNIVERSIDAD DE MÁLAGA

con la siguiente ecuación:

$$correctw = (Q_L[j] * d_j + D[j] * q_{hj+1}) * 10^{-\delta+1}$$

Además, el dígito x_j es sumado a la posición $10^{-\delta+2}$ del residuo $w[j]$, la posición es definida por el retardo online (δ) y la inicialización $w[0] = x/100$

5.2.3. Módulo de Suma de Residuo

El módulo de suma de residuo w es el mismo que el usado en la sección 4.2.1. Como muestra la Figura 5.2, es necesario dos módulos de suma para obtener el residuo.

Con el primer módulo se obtiene $v[j]$ como resultado de la suma: $v[j] = 10w[j] - q_{Hj+1}(5D[j]) - correctv \cdot 10^{-\delta+1}$. Con dicho resultado obtenemos q_{Lj+1} mediante la función de selección $SEL_L(\widehat{v[j]}, \widehat{D[j+\delta-1]})$ definida en la sección 5.2.1.

Con el segundo módulo y el posterior desplazamiento de 4 bits a la izquierda se obtiene el residuo para el siguiente ciclo $w[j+1]$. El resultado lo obtenemos mediante la suma: $W[j+1] = v[j] - q_{Lj+1}d - correctw \cdot 10^{-\delta+1}$.

5.3. Resultados Experimentales

En esta sección se presentan los resultados de la división decimal online donde la arquitectura presentada en este apartado se ha modelado y verificado a nivel RTL con Verilog. También, se ha sintetizado usando Synopsis Design Compiler y la libería de celdas TSMC's tcbn65gplus 65 nm CMOS. Todo el entorno y parámetros de proceso se fijaron a las condiciones normales o típicas.

Debido a que no existe en la literatura un divisor decimal online y siguiendo los resultados del capítulo de la multiplicación decimal online, descartamos la implementación y estudio de la división decimal online con especulación. La Tabla 5.2 muestra los resultados en área y en retardo de la división online decimal propuesta en este capítulo.

En comparación con los resultados de la multiplicación decimal online presentados en la sección 4.3, el retardo aumenta debido a la complejidad de la división, ya que se duplica el módulo de suma y se utilizan funciones de selección más complejas y su LUT.

UNIVERSIDAD DE MÁLAGA

Diseño	Retardo (ns)	Area ($\mu\ m^2$)	Retardo MSD (ns)
División Online propuesta	2.65	33288	10.6

Tabla 5.2: Resultados de la división online

5.4. Conclusiones del Capítulo

En este capítulo se ha presentado un algoritmo para la división decimal online basado en el trabajo presentado en [20]. Los principales puntos de diseño del algoritmo propuesto son:

- Constantes de selección para la obtención del cociente usando la codificación RBCD.
- Módulo suma de residuo (6 dígitos RBCD) que acumula el residuo generado en los anteriores ciclos.
- Módulo de corrección que contiene una multiplicación vector por digito para obtener el error producido en los ciclos anteriores debido a la entrada de nuevos dígitos de entrada cada ciclo.
- Cálculo del cociente divido en q_{Hj+1} y q_{Lj+1}, definiendo una función de selección para obtener cada uno de los dígitos. Esta separación del cociente optimiza el retardo del algoritmo ya que paraleliza la computación de las funciones de selección.

Se han presentado los resultados obtenidos mediante la simulación del algoritmo, obteniendo un retardo y un área que debido a la complejidad de la operación, son relativamente mayores que la multiplicación haciendo que los resultados obtenidos sean acordes con el estudio teórico.

UNIVERSIDAD DE MÁLAGA

6 Conclusiones y Trabajo Futuro

Debido al amplio espectro de aplicaciones comerciales como son el análisis financiero, banca, cálculo de tasas y operaciones contables se realizan utilizando aritmética binaria, que introduce un error de precisión al convertir un número decimal a binario y viceversa, se publicó el estándar IEEE 754-2008 con el que se ha renovado el interés por la aritmética decimal y la búsqueda de nuevos algoritmos y codificaciones que puedan facilitar las operaciones decimales. Debido a ello, los fabricantes de procesadores como IBM, SparcX o Fujitsu han incluido recientemente una arquitectura decimal en punto flotante.

En el contexto de aritmética decimal, hemos propuesto una solución para realizar operaciones utilizando la aritmética online con una codificación redundante y aprovechar las ventajas de ambas aritméticas para implementar una unidad aritmético-lógica ayudando a la paralelización de operaciones aritméticas debido a la obtención del dígito mas significativo primero y la reducción del área de la arquitectura.

La primera operación propuesta en esta tesis y la más básica es la suma decimal online. Con un estudio exhaustivo de modelo de suma y distintas codificaciones se obtiene un modelo y una codificación (RBCD) óptimos que serán utilizados para la arquitectura de la suma decimal online (olDFA) propuesta. A su vez se propone una versión del olDFA segmentada con tres etapas (olDFA$_p$) con el objetivo de reducir el tiempo de ciclo. También se ha tratado el stream de datos, proponiendo una solución para obtener el máximo throughput teórico posible en un sumador online. La inserción de dos puertas estratégicamente colocadas en el diseño, hace posible el intervalo de iniciación mínimo con un coste hardware despreciable.

UNIVERSIDAD DE MÁLAGA

Además proponemos otra arquitectura online basada en la arquitectura más utilizada para la suma decimal obteniendo mejores resultados para la arquitectura olDFA tanto en área como en tiempo de ciclo. Basándonos en el olDFA, se propone un método para construir árboles de sumadores basados en olDFAs y olDFA$_p$s para a minimizar recursos hardware y, por consiguiente, consumo de energía en sumas decimales online multioperando.

Terminando con la suma decimal online, debido a que el uso de algunos de los formatos decimales permiten la optimización de algoritmos decimales, se propone un método para diseñar sumadores decimales online multiformato y multioperando. En primer lugar, el sumador decimal online multiformato (olDFAMformat) ha sido abordado con dos estrategias de diseño diferentes: la primera, se ha basado en un olDFA con una etapa de etapa de conversión; y la segunda ha sido basada en el diseño específico para lo que se ha modificado la arquitectura interna del olDFA. La comparación de las estrategias ha mostrado que la primera es la mejor opción cuando cada entrada del sumador puede ser representada por más de dos códigos, mientras que si las entradas sólo pueden ser representadas por dos códigos, la segunda opción sería la mejor con respecto al retardo.

La siguiente contribución es la multiplicación decimal online, donde se propone un algoritmo usando la codificación RBCD y dos arquitecturas comparando dos formas distintas de implementación del algoritmo. La arquitectura que implementa la multiplicación decimal online sin especulación es la arquitectura más óptima ya que tiene la mejor relación entre coste computacional y coste hardware. Se ha diseñado un multiplicador decimal online RTL de 16x16 dígitos, y se ha insertado en un sistema online. Por motivos de comparación, también hemos implementado el mismo sistema pero sustituyendo nuestro multiplicador online por un multiplicador decimal paralelo rápido, obteniendo un incremento en velocidad del 22.5 % para obtener el MSD y a su vez reduciendo el área un 60 %.

La última operación propuesta es la división decimal online diseñando un algoritmo optimizando tiempo de ciclo y área implementando un módulo de constantes de selección, un módulo de corrección y separando el valor del cociente en dos variables. Comparando con la multiplicación se duplica el retardo debido a la utilización de la LUT utilizada para las constantes de selección y de la duplicación del módulo de suma necesario para obtener el residuo acumulado en cada ciclo.

Como líneas de trabajos futuras, creemos que serían interesantes los siguientes trabajos:

- Estudiar cómo se adaptaría el algoritmo para realizar la raíz cuadrada de-

UNIVERSIDAD DE MÁLAGA

cimal online, además de proponer la/s arquitectura/s más óptima/s.

- Estudio del diseño de una unidad combinada de raíz cuadrada y división.
- Analizar el diseño para realizar la comparación online de dos números decimales.
- Analizar igualmente el diseño que resultaría si se quisiera realizar el valor absoluto de un número decimal.
- Indagar en el campo financiero otras funciones decimales usuales como la función exponencial por ejemplo.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Apéndice A
Estudio de Modelos de Suma

En el capítulo 2 se presentó el sumador decimal sobre el que se ha basado esta tesis. Y en la sección 3.2 se mencionó que existen muchos códigos decimales de 4 bits que cumplían la condición para tener la redundancia suficiente para prevenir una propagación de acarreo. Hemos estudiado algunos de esos códigos y el resultado de dicho estudio lo presentamos a lo largo de este apéndice.

En concreto, estudiamos diferentes códigos decimales redundantes en los que el bit más significativo tiene peso negativo y el resto tienen un peso positivo. Además analizamos esquemas de descomposición diferentes al seguido en el sumador DSSD [16] pero que siguen manteniendo sus mismas características. Deducimos todas las codificaciones (apartado A.1) y los diseños de la descomposición (apartado A.2) que cumplen con estos requisitos. Dichos diseños son analizados y comparados con el sumador decimal redundante DSSD. Además, estudiaremos los problemas debido al rango de representación y proporcionaremos diferentes soluciones cuando sea posible.

A.1. Codificaciones Redundantes

Consideramos una codificación general dada por la cuaterna $\Omega = (N, a, b, c)$ dónde cada componente representa un número entero. El primer supuesto que tenemos que asumir es el signo de cada uno de ellos, y puesto que buscamos una representación en Complemento a dos, tenemos:

UNIVERSIDAD DE MÁLAGA

i. N es un número entero negativo, mientras que los valores a, b, c son enteros positivos, y todos ellos no nulos: N $\in Z^-$; a, b, c $\in Z^+$.

Por otro lado, como queremos obtener una codificación decimal, cada componente deberá cumplir:

ii. $|N| < 9; |a| < 9; |b| < 9; |c| < 9$.

Teniendo en cuenta estas primeras condiciones, el número de posibilidades que se nos presenta inicialmente es de $9^4 = 6561$, por lo que tendremos que afinar mucho en la búsqueda de requisitos para reducir el trabajo. Por ejemplo, para no repetir el número de codificaciones estudiadas, consideraremos la siguiente relación de orden entre componentes (positivos):

iii. $a \geq b \geq c$.

Observamos que del resultado de la suma debe poder extraerse un acarreo, así como soportar otro acarreo procedente de una suma anterior si fuese necesario, y sin que éste genere uno nuevo a su vez. Es decir, si x, y $\in [-\alpha, \beta]$ ha de verificarse:

$x + y = 10 \cdot \{-1,0,1\} + q \Rightarrow q \in [-\alpha + 1, \beta - 1] = B$

La definición del conjunto B nos asegura que el nuevo acarreo no produzca otro exceso de rango a su vez.

iv. Si t, s $\in S : s < -\alpha + 1$, entonces t = -10+i, para i $\in B$

v. Si t, s $\in S : \beta - 1 < s$, entonces t = 10+i, para i $\in B$

Es decir, con N= -8 necesitamos representar el dígito -7, por ello, el valor de a, b o c tiene que ser igual a 1. Para obtener la cuaterna correspondiente declaramos:

v. c=1

Hay que tener en cuenta que los valores de a y b tienen que estar cercanos a c para obtener una cuaterna con rango válido. Por ejemplo, teniendo la cuaterna Ω=(-8,3,3,1) se obtiene el rango erróneo {-8, -7, -5, -4, -2, -1, 0, 1, 3, 4, 6, 7}.

UNIVERSIDAD DE MÁLAGA

N	0
N+c	c
N+b	b
N+b+c	b+c
N+a	a
N+a+c	a+c
N+a+b	a+b
N+a+b+c	a+b+c

Por ello se deduce una restricción para controlar este problema. Teniendo en cuenta la condición iii., podemos obtener todas las representaciones numéricas de la siguiente forma:

Sabemos que para conseguir un rango con sentido, el conjunto tiene que estar formado por números enteros consecutivos. Por ejemplo, fijándonos en la variable b, tiene que ocurrir que $|b-c| \leq 1$ ó $|(N+b)-(N+c)| \leq 1$. De ahí, y aplicando vi. se tienen las mismas restricciones para b de ambas inecuaciones:

$$|b-c| \leq 1 \Rightarrow^{vi} |b-1| \leq 1 \Rightarrow$$

$$-1 \leq b-1 \leq 1 \rightarrow \begin{cases} -1 \leq b-1 \Rightarrow 0 \leq b \\ b-1 \leq 1 \Rightarrow b \leq 2 \end{cases}$$

$$|(N+b)-(N+c)| \leq 1 \Rightarrow |(N+b)-(N+1)| \leq 1 \Rightarrow |b-1| \leq 1$$

Considerando la condición i. y esta nueva condición, tenemos:

vii. b$\in \{1,2\}$

Para obtener la mínima redundancia, decidimos reducir el rango de dígitos de N a:

viii. N $\in \{-4,-5,-6,-7,-8,-9\}$

Usando todas estas condiciones, obtenemos las siguientes 12 codificaciones válidas:

$$\begin{array}{ll} \Omega_1 = (-4,4,2,1) & \Omega_7 = (-6,2,2,1) \\ \Omega_2 = (-4,5,2,1) & \Omega_8 = (-6,3,2,1) \\ \Omega_3 = (-4,6,2,1) & \Omega_9 = (-6,4,2,1) \\ \Omega_4 = (-5,3,2,1) & \Omega_{10} = (-7,3,2,1) \\ \Omega_5 = (-5,4,2,1) & \Omega_{11} = (-7,4,2,1) \\ \Omega_6 = (-6,3,2,1) & \Omega_{12} = (-8,4,2,1) \end{array}$$

UNIVERSIDAD DE MÁLAGA

A.2. Modelos de Suma

El diseño en el que nos estamos basando [16] descompone los dos sumandos en los conjuntos $v_i = \{y_i^1, x_i^0, y_i^0\}$ y $u_i = \{X_i^3, Y_i^3, x_i^2, y_i^2, x_i^1\}$, y éste último se descompone a su vez en un acarreo (t_{i+1}^0, T_{i+1}^0) y en un residuo que queda codificado mediante el conjunto z_i (Z_i^2, z_i^2, Z_i^1). En resumen, se establecía lo siguiente:

$||u|| = ||z_i|| + 10 \cdot ||t_{i+1}||$ con

$||z_i|| \in \{-6, -4, -2, 0, 2\}$, $||t_{i+1}|| \in \{-1, 0, 1\}$

$||v_i|| \in [0, 4]$

$z_i \Rightarrow (Z_i^2, z_i^2, Z_i^1)$

$v_i \Rightarrow (v_i^2, v_i^1, V_i^0)$

$t_{i+1} \Rightarrow (t_{i+1}^0, T_{i+1}^0)$

Observamos que se han elegido los bits más significativos para el primer conjunto y los menos significativos para el segundo. De esta forma, el acarreo puede ser extraído en aquél grupo con valores más grandes. Por tanto, es lógico adjudicar a z_i bits de peso mayor, y dejar en v_i a los de menor peso.

Por otro lado, no hay que olvidar que cada codificación hallada en la sección anterior tiene su propio rango de valores, por lo que las elecciones de z_i y v_i que resulten válidas podrán ser distintas en cada caso.

A.3. Descomposición de v

De acuerdo con el grupo v_i compuesto por los bits y_i^1, x_i^0, y_i^0 obtenemos diferentes combinaciones de descomposición en negabits y posibits. Las Tablas A.1 y A.2 reúnen todas las posibles combinaciones, donde las mayúsculas son negabits y las minúsculas son posibits. Los guiones (—-) representan combinaciones no válidas.

	V_i^0	$v_i^1\ V_i^0$	$V_i^1\ V_i^0$	$v_i^1\ v_i^1\ V_i^0$	$v_i^1\ V_i^1\ V_i^0$
	—-	$v_i^1\ V_i^0$	—-	$v_i^1\ V_i^0\ v_i^1$	$v_i^1\ V_i^0\ V_i^1$
v_i^2	—-	$v_i^2\ v_i^1\ V_i^0$	$v_i^2\ V_i^1\ V_i^0$	—-	—-
V_i^2	—-	$V_i^2\ v_i^1\ V_i^0$	—-	—-	—-
$v_i^2\ v_i^2$	—-	—-	—-	—-	—-
$V_i^2\ V_i^2$	—-	—-	—-	—-	—-

Tabla A.1: Combinaciones válidas para el grupo v

UNIVERSIDAD DE MÁLAGA

		v_i^1	V_i^1	v_i^2	V_i^2
$V_i^0\ v_i^0$	—	$v_i^1\ V_i^0\ v_i^0$	—	$v_i^2\ V_i^0\ v_i^0$	—

Tabla A.2: Resto de combinaciones válidas para el grupo v

A.4. Descomposición de z

La Tabla A.3 muestra las combinaciones para el grupo z_i. Las celdas eliminadas corresponden a combinaciones en las cuales alguna representación numérica no es posible o el número de bits se excede o es repetido. Por ejemplo, teniendo en cuenta el transfer $T_{i+1}^0\ t_{i+1}^0$, usando la combinación $z_i^2\ Z_i^1$, algunas representaciones pares no se pueden obtener: -12, -10, -6, -2, 0, 4, 8, 10 y 14. Una combinación válida es $z_i^2\ Z_i^1\ Z_i^1$ con el rango: {-14, -12, -10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10, 12, 14}. La Tabla A.4 muestra todas las combinaciones posibles para el grupo z_i así como las numeraciones que se usarán para diferenciarlas.

	$z_i^2\ z_i^1$	$z_i^2\ Z_i^1$	$Z_i^2\ z_i^1$	$Z_i^2\ Z_i^1$
	—	—	—	—
z_i^1	$z_i^2\ z_i^1\ z_i^1$	$z_i^2\ Z_i^1\ z_i^1$	$Z_i^2\ z_i^1\ z_i^1$	$Z_i^2\ Z_i^1\ z_i^1$
Z_i^1	—	$z_i^2\ Z_i^1\ Z_i^1$	—	$Z_i^2\ Z_i^1\ Z_i^1$
z_i^2	$z_i^2\ z_i^2\ z_i^1$	$z_i^2\ z_i^2\ Z_i^1$	—	—
Z_i^2	$Z_i^2\ z_i^2\ z_i^1$	$z_i^2\ Z_i^2\ Z_i^1$	$Z_i^2\ Z_i^2\ z_i^1$	$Z_i^2\ Z_i^2\ Z_i^1$

Tabla A.3: Combinaciones válidas para el grupo z

1z.	$t^0\ T^0\ z^2\ z^1\ z^1$
2z.	$t^0\ T^0\ z^2\ Z^1\ z^1$
3z.	$t^0\ T^0\ z^2\ z^2\ z^1$
4z.	$t^0\ T^0\ Z^2\ z^1\ z^1$
5z.	$t^0\ T^0\ z^2\ Z^1\ Z^1$
6z.	$t^0\ T^0\ z^2\ z^2\ Z^1$
7z.	$t^0\ T^0\ Z^2\ z^1\ z^1$
8z.	$t^0\ T^0\ Z^2\ Z^2\ z^1$
9z.	$t^0\ T^0\ Z^2\ Z^1\ z^1$
10z.	$t^0\ T^0\ Z^2\ Z^1\ Z^1$
11z.	$t^0\ T^0\ Z^2\ z^2\ Z^1$
12z.	$t^0\ T^0\ Z^2\ Z^2\ Z^1$

Tabla A.4: Numeración de las descomposiciones de u_i

UNIVERSIDAD DE MÁLAGA

A.5. Descomposición Válida para cada Codificación

En esta sección se estudia el rango de la descomposición para cada codificación obtenida en la sección A.1. Por ejemplo, el modelo basado en el presentado en [16] usa la codificación -8421 y realiza la descomposción de las entradas con el rango de los grupos v_i y u_i descritos anteriormente. Por ello los rangos de este modelo son:

v_i={2,1,1}

$||v_i||$={0,1,2,3,4}

u_i={-8,-8,4,4,2}

$||u_i||$={-16,-14,-12,-10,-8,-6,-4,0,2,4,6,8,10}

El grupo u_i es codificado por los transfers T^0_{i+1}, t^0_{i+1} y el rango de z_i. Por ello, los rangos de u_i yv_i son:

T^0_{i+1}, $t^0_{i+1} \in \{-1, 0, 1\}$

$z_i \Rightarrow (Z^2_i, z^2_i, Z^1_i) \Rightarrow ||z_i|| = \{-6, -4, -2, 0, 2, 4\}$

$v_i \Rightarrow (v^2_i, v^1_i, v^0_i) \Rightarrow ||v_i|| = \{-1, 0, 1, 2, 3, 4, 5, 6\}$

Por otro lado, el rango de $||u_i||$ es:

$||u_i||=||z_i|| + 10 \cdot ||t_{i+1}|| =$

$\{-16, -14, -12, -10, -8, -6, -4, 0, 2, 4, 6, 8, 10, 12, 14, 16\}$

Analizamos los rangos para el resto de posibles soluciones obtenidas en la sección previa para obtener las codificaciones válidas para cada combinación. Para obtener las codificaciones válidas para cada combinación de v_i analizamos las codificaciones que acaban en XX21 y la codificación -6311. Comparamos los rangos de las entradas $y^1_i y^0_i x^0_i$ para cada codificación con el rango de cada combinación de v_i para descartar las combinaciones no válidas que no pueden representar el rango de las entradas. La Tabla A.5 muestra el resumen de nuestro análisis.

	Combination	Digit set	Valid Codifications
1	$v^2_i\ v^1_i\ V^0_i$	{-1,0,1,2,3,4,5,6}	All
2	$v^2_i\ V^1_i\ V^0_i$	{-3,-2,-1,0,1,2,3,4}	All
3	$v^1_i\ v^1_i\ V^0_i$	{-1,0,1,2,3,4}	All
4	$v^1_i\ V^0_i\ v^0_i$	{-1,0,1,2,3}	-6311

Tabla A.5: Resumen del análisis de v_i con todas las codificaciones

UNIVERSIDAD DE MÁLAGA

En esta sección analizamos la viabilidad de obtener un diseño de suma mejorado con un resultado RBCD. Combinando las Tablas A.5, A.6 y A.7 obtenemos varios modelos a estudiar. Cada modelo tiene una arquitectura diferente.

Numeración z	-4421	-4521	-4621	-5321	-5421	-6221
1z. $t^0\ T^0\ z^2\ z^1\ z^1$	x	x	x	x	x	
2z. $t^0\ T^0\ z^2\ Z^1\ z^1$	x	x	x	x	x	x
3z. $t^0\ T^0\ z^2\ z^2\ z^1$	x	x	x	x		
4z. $t^0\ T^0\ Z^2\ z^1\ z^1$	x	x	x	x	x	x
5z. $t^0\ T^0\ z^2\ Z^1\ Z^1$	x	x	x	x	x	x
6z. $t^0\ T^0\ z^2\ z^2\ Z^1$	x	x	x	x	x	x
7z. $t^0\ T^0\ Z^2\ z^1\ z^1$	x	x	x	x	x	x
8z. $t^0\ T^0\ Z^2\ Z^2\ z^1$	x	x		x	x	x
9z. $t^0\ T^0\ Z^2\ Z^1\ z^1$	x	x		x	x	x
10z. $t^0\ T^0\ Z^2\ Z^1\ Z^1$	x			x	x	x
11z. $t^0\ T^0\ Z^2\ z^2\ Z^1$	x	x	x	x	x	x
12z. $t^0\ T^0\ Z^2\ Z^2\ Z^1$	x			x	x	x

Tabla A.6: Codificaciones válidas para cada descomposición de u_i

Numeración z	-6311	-6321	-6421	-7321	-7421	8421
1z. $t^0\ T^0\ z^2\ z^1\ z^1$						
2z. $t^0\ T^0\ z^2\ Z^1\ z^1$	x	x	x			
3z. $t^0\ T^0\ z^2\ z^2\ z^1$						
4z. $t^0\ T^0\ Z^2\ z^1\ z^1$	x	x	x	x	x	
5z. $t^0\ T^0\ z^2\ Z^1\ Z^1$	x	x	x	x	x	
6z. $t^0\ T^0\ z^2\ z^2\ Z^1$	x	x	x			
7z. $t^0\ T^0\ Z^2\ z^1\ z^1$	x	x	x	x	x	
8z. $t^0\ T^0\ Z^2\ Z^2\ z^1$	x	x	x	x	x	x
9z. $t^0\ T^0\ Z^2\ Z^1\ z^1$	x	x	x	x	x	x
10z. $t^0\ T^0\ Z^2\ Z^1\ Z^1$	x	x	x	x	x	x
11z. $t^0\ T^0\ Z^2\ z^2\ Z^1$	x	x	x	x	x	x
12z. $t^0\ T^0\ Z^2\ Z^2\ Z^1$	x	x	x	x	x	x

Tabla A.7: Resto de codificaciones válidas para cada descomposición de u_i

Se analiza los niveles lógicos de las combinaciones $z - v$ en la Tabla A.8 sin tener en cuenta el módulo de descomposición que se estudiará más adelante.

Nuestro modelo de referencia es el modelo presentado en [16] con un camino crítico de 9 niveles lógicos (3 Full Adders, un half adder y una puerta NOR). Nosotros fijamos el camino crítico como el límite para estudiar en profundidad el modelo. Se descartan todos los modelos con un camino crítico mayor que el modelo de referencia y se analiza la validación del resto de modelos con cada codificación de la sección A.1.

UNIVERSIDAD DE MÁLAGA

	1. $v^2\ v^1\ V^0$	2. $v^2\ V^1\ V^0$	3. $v^1\ v^1\ V^0$	4. $v^1\ V^0\ v^0$
1z.	9n;3F2HA,1XOR	9n;3F2HA,1XOR	9n;3F3HA,1XOR	12n;3F8HA
2z.	9n;3F2HA,1XOR	9n;3F2HA,1XOR	9n;3F3HA,1XOR	11n;3F3HA,1XOR
3z.	10n;3F4HA,1XOR	9n;3F1HA,1XOR	9n;3F2HA,1XOR	11n;3F3HA,1XOR
4z.	9n;3F2HA,1XOR	9n;3F1HA,1XOR	9n;3F2HA,1XOR	11n;3F3HA,1XOR
5z.	9n;3F2HA,1XOR	9n;3F2HA,1XOR	9n;3F3HA,1XOR	11n;3F4HA,1XOR
6z.	9n;3F1HA,1XOR	9n;3F2HA,1XOR	9n;3F2HA,1XOR	11n;3F3HA,1XOR
7z.	9n;3F2HA,1XOR	9n;3F2HA,1XOR	9n;3F3HA,1XOR	9n;3F3HA,1XOR
8z.	—	9n;3F1HA,1XOR	9n;3F2HA,1XOR	11n;3F3HA,1XOR
9z.	9n;3F2HA,1XOR	9n;3F2HA,1XOR	9n;4FA	11n;4F1HA
10z.	9n;3F2HA,1XOR	9n;3F2HA,1XOR	9n;3F3HA,1XOR	12n;3F8HA
11z.	9n;3F1HA,1XOR	—	9n;3F2HA,1XOR	11n;3F3HA,1XOR
12z.	9n;3F1HA,1XOR	9n;3F2HA,1XOR	9n;3F2HA,1XOR	11n;3F3HA,1XOR

Tabla A.8: Camino crítico para cada modelo sin el módulo de descomposición

Cada celda de la Tabla A.8 detalla el número de niveles y el número de Full Adders (FA), Half Adders (HA) y puertas lógicas necesarias para obtener un diseño con el mínimo de área y lo más eficiente posible. En la Tabla A.8, mFnHA corresponde a m Full Adders y n Half Adders. No es posible obtener un diseño válido para obtener un resultado en Complemento a dos usando los modelos 8-11 y 11-2 debido a que no hay una combinación de posibits y negabits con el cual obtener un resultado en Complemento a dos.

Para el resto de modelos, obtenemos un diseño válido con un resultado en C2. Como se muestra en la Tabla A.8 hay muchas combinaciones $z-v$ que tienen más niveles lógicos o área que el modelo base [16], que coincide con el modelo 11-1 obtenido. Sin embargo hay siete combinaciones que tienen, al menos, las mismas características:

- 3-2: $t^0\ T^0\ z^2\ z^2\ z^1$; $v^2\ V^1\ V^0 \Rightarrow$ 9lvl,3FA,1HA,1XOR
- 4-2: $t^0\ T^0\ Z^2\ z^1\ z^1$; $v^2\ V^1\ V^0 \Rightarrow$ 9lvl,3FA,1HA,1XOR
- 6-1: $t^0\ T^0\ z^2\ z^2\ Z^1$; $v^2\ v^1\ V^0 \Rightarrow$ 9lvl,3FA,1HA,1XOR
- 8-2: $t^0\ T^0\ Z^2\ Z^2\ z^1$; $v^2\ V^1\ V^0 \Rightarrow$ 9lvl,3FA,1HA,1XOR
- 9-3: $t^0\ T^0\ Z^2\ Z^1\ z^1$; $v^1\ v^1\ V^0 \Rightarrow$ 9lvl,4FA
- 11-1*:$t^0\ T^0\ Z^2\ z^2\ Z^1$; $v^2\ v^1\ V^0 \Rightarrow$ 9lvl,3FA,1HA,1XOR
- 12-1:$t^0\ T^0\ Z^2\ Z^2\ Z^1$; $v^2\ v^1\ V^0 \Rightarrow$ 9lvl,3FA,1HA,1XOR

UNIVERSIDAD DE MÁLAGA

Modelos	-4421	-4521	-4621	-5321	-5421	-6221
3-2	x*	x*	x*	x*		
4-2	x	x*	x*	x*	x*	x
6-1	x*	x*	x*	x*	x*	x*
8-2	x	x		x	x	x
9-3	x	x		x	x	x
11-1	x	x	x	x	x	x
12-1	x*			x*	x*	x*

Tabla A.9: Codificaciones válidas para cada modelo

La columna 4 de la Tabla A.8, la cual es válida sólo para la codificación -6311, es descartada debido a que los niveles lógicos y área son mayores que la celda 11-1, la cual corresponde al modelo base [16] y es mostrado en la figura A.1. La figura A.2 muestra el modelo 8-2 y el camino crítico va por 2 FA, 1 HA y una puerta XOR. El modelo 3-2 tiene la misma arquitectura que el modelo 8-2 con la diferencia del signo de los bits con peso 2 del grupo u_i, la figura A.3 muestra el modelo 9-3 donde el camino crítico va por 3 FA.

Para obtener las codificaciones válidas para cada modelo, combinamos las Tablas A.6, A.7 y A.5. Las Tablas A.9 y A.10 muestran las codificaciones válidas para los siete modelos descritos anteriormente. Basándonos en las Tablas A.9 y A.10 analizamos la descomposición para cada modelo compatible y codificación (marcado con una x) para obtener el camino crítico y seleccionar los modelos a descartar (marcados con un *) debido a que el camino crítico de la descomposición es mayor que 13 puertas lógicas obtenidas en el modelo base [16].

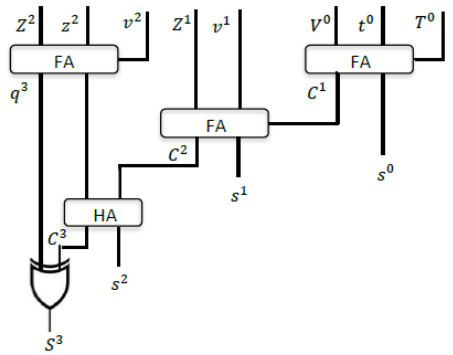


Figura A.1: Modelo 11-1.

UNIVERSIDAD DE MÁLAGA

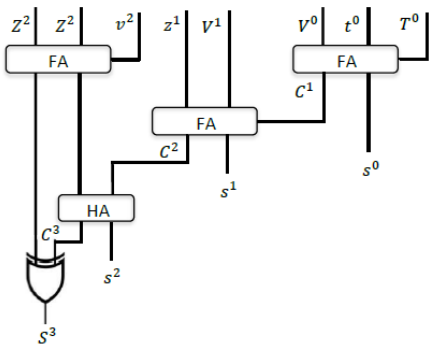


Figura A.2: Modelo 8-2.

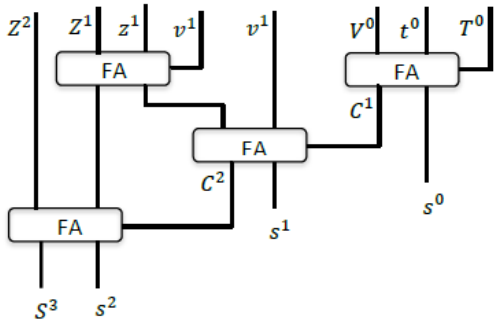


Figura A.3: Modelo 9-3.

Modelos	-6311	-6321	-6421	-7321	-7421	-8421
3-2						
4-2	x*	x*	x	x*	x*	
6-1	x*	x*				
8-2	x	x	x	x	x	x
9-3	x	x	x	x	x	x
11-1	x	x	x	x	x	x
12-1	x*	x*	x*	x*	x*	x*

Tabla A.10: Resto de codificaciones válidas para cada modelo

UNIVERSIDAD DE MÁLAGA

A.6. Desbordamiento

Para representar los dígitos {4,6,8} en la descomposición del grupo u_i, se necesita un transfer para la siguiente suma y así obtener un resultado en RBCD con el rango {-7,...,0,...,7} debido a que el grupo v_i tiene un rango {0,...,4} para codificaciones XX21. Se analiza el desbordamiento producido en algunos modelos como consecuencia de un exceso de posibits o negabits en la descomposición. El modelo 3-2 (t^0 T^0 z^2 z^2 z^1; v^2 V^1 V^0) es descompuesto usando dos posibits con peso dos (z_1^2 z_2^2) se obtiene un valor de 8 cuando los valores de ambos es 1, por lo tanto, descartamos este modelo debido a que el desbordamiento producido genera un incremento del camino crítico y la descomposición alternativa no es posible. El modelo 4-2 (t^0 T^0 Z^2 z^1 z^1; v^2 V^1 V^0) y el modelo 6-1 (t^0 T^0 z^2 z^2 Z^1; v^2 v^1 V^0) son descartados por las mismas razones. Por otro lado, el grupo u_i del modelo 12-1 (t^0 T^0 Z^2 Z^2 Z^1; v^2 v^1 V^0) es descompuesto usando 3 negabits y, por lo tanto, el camino crítico se incrementa y el modelo es descartado.

A.7. Resultados Experimentales y Comparación

En esta sección se analiza el rendimiento de las arquitecturas propuestas en este apéndice. Han sido modeladas en Verilog-HDL y verificadas usando ModelSim 6.0 para cada posible combinación en la entrada. También, hemos sintetizado los diseños usando Synopsis Design Compiler (DC) y la librería de celda TSMC's tcbn65gplus 65 nm CMOS standard

La Tabla A.11 muestra los resultado de los diseños donde, como se ha mencionado anteriormente, el modelo base es el modelo 11-1 con la codificación -8421 y el resto son propuestos en este capítulo. Usando la codificación -8421, se obtiene una reducción del retardo (4,2 %) y área (5,9 %) usando el modelo 8-2 presentado en este capítulo con respecto al modelo base 11-1 presentado en [16]. Por otro lado, usando entradas codificadas en -4421, el modelo 8-2 tiene los mejores resultados de delay con una reducción de retardo (9,7 %) y área (11,4 %) comparado con el modelo base. Se puede observar en los resultados que el retardo y el área del modelo 8-2 es menor que el de otros modelos presentados, teniendo una reducción del retardo del 14 %. Aunque el área se ha incrementado un 4 % con respecto al modelo 11-1 con codificación -4421. Los resultados del modelo 9-3 muestran que el modelo incrementa el retardo y el área con respecto al modelo base. Por ello se descarta el modelo 9-3 como sumador paralelo mejorado.

UNIVERSIDAD DE MÁLAGA

Diseño	Retardo (ns)	Área (μm^2)
Modelo 8-2 Codificación -4421	0.306	386
Modelo 8-2 Codificación -8421	0.324	411
Modelo 9-3 Codificación -4421	0.354	437
Modelo 9-3 Codificación -8421	0.353	445
Modelo 11-1 Codificación -4421	0.356	368
Modelo 11-1 Codificación -8421 [5]	0.339	436

Tabla A.11: Resultados Experimentales

A.8. Conclusión

Basándonos en el sumador decimal paralelo propuesto en [16], este apéndice ha presentado un estudio de todas las posibles combinaciones entre todas las codificaciones decimales redundantes y diseños de sumadores decimales paralelos, obteniendo un modelo (Modelo 8-2) de suma redundante que reduce el retardo y el área con respecto el modelo base.

UNIVERSIDAD DE MÁLAGA

Apéndice B
Funciones de Conversión a RBCD

Describimos todas las funciones de conversión de otro código estudiado en el texto principal a RBCD

- Conversión RBCD-$_{7421}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \cdot (\overline{x_i^2 \cdot x_i^1 \cdot x_i^0}) \\ y_i^2 &= \overline{X_i^3} \cdot x_i^2 + x_i^3 \cdot (x_i^2 \oplus (x_i^1 \cdot x_i^0)) \\ y_i^1 &= \overline{X_i^3} \cdot x_i^1 + X_i^3 \cdot (x_i^1 \oplus x_i^0) \\ y_i^0 &= X_i^3 \oplus x_i^0 \end{aligned} \tag{B.1}$$

- Conversión RBCD-$_{7321}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \\ y_i^2 &= x_i^2 \cdot (x_i^1 + x_i^0) + X_i^3 \cdot (x_i^1 \cdot x_i^0 + x_i^2) \\ y_i^1 &= \overline{X_i^3} \cdot (x_i^1 \cdot x_i^0 + \overline{x_i^2} \cdot x_i^1 + x_i^2 \cdot \overline{x_i^1} \cdot \overline{x_i^0}) + \\ &\quad X_i^3 \cdot (x_i^2 \cdot x_i^1 + \overline{x_i^2} \cdot \overline{x_i^1} \cdot x_i^0 + \overline{x_i^2} \cdot x_i^1 \cdot \overline{x_i^0}) \\ y_i^0 &= (X_i^3 \oplus x_i^2) \oplus x_i^0 \end{aligned} \tag{B.2}$$

UNIVERSIDAD DE MÁLAGA

- Conversión RBCD$_{-6421}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \cdot (\overline{x_i^2 \cdot x_i^1}) \\ y_i^2 &= \overline{X_i^3} \cdot x_i^2 + x_i^3 \cdot (x_i^2 \oplus x_i^1) \\ y_i^1 &= \overline{X_i^3} \cdot x_i^1 + X_i^3 \cdot (x_i^1) \\ y_i^0 &= X_i^3 \end{aligned} \tag{B.3}$$

- Conversión RBCD$_{-6321}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \cdot (\overline{x_i^2 \cdot x_i^1 \cdot x_i^0}) \\ y_i^2 &= \overline{X_i^3} \cdot x_i^2 \cdot (x_i^1 + x_i^0) + x_i^3 \oplus (x_i^1 + x_i^1 \cdot \overline{x_i^0}) \\ y_i^1 &= X_i^3 \oplus (x_i^1 \cdot x_i^0 + \overline{x_i^2} \cdot x_i^1 + x_i^2 \overline{x_i^1 x_i^0}) \\ y_i^0 &= x_i^2 \oplus x_i^0 \end{aligned} \tag{B.4}$$

- Conversión RBCD$_{-6221}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \\ y_i^2 &= \overline{X_i^3} \cdot x_i^2 \cdot x_i^1 + x_i^3 \cdot x_i^1 + x_i^3 \cdot x_i^2 \\ y_i^1 &= X_i^3 \oplus (x_i^2 \oplus x_i^1) \\ y_i^0 &= x_i^0 \end{aligned} \tag{B.5}$$

- Conversión RBCD$_{-5421}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \cdot (\overline{x_i^2} + \overline{x_i^1} \cdot \overline{x_i^0}) \\ y_i^2 &= \overline{X_i^3} \cdot x_i^2 + X_i^3 \cdot ((x_i^1 + x_i^0) \oplus x_i^2) \\ y_i^1 &= \overline{X_i^3} \cdot x_i^1 + X_i^3 \cdot (\overline{x_i^1 \oplus x_i^0}) \\ y_i^0 &= X_i^3 \oplus x_i^0 \end{aligned} \tag{B.6}$$

- Conversión RBCD$_{-5321}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \cdot (\overline{x_i^2 \cdot x_i^1}) \\ y_i^2 &= (X_i^3 \oplus x_i^2) \cdot (x_i^1 + x_i^0) + X_i^3 \cdot x_i^2 \cdot \overline{x_i^1} \\ y_i^1 &= (X_i^3 \oplus x_i^2) \cdot (x_i^1 \oplus x_i^0) + \overline{X_i^3} \cdot \overline{x_i^2} \cdot x_i^1 + \\ &\quad X_i^3 \cdot x_i^2 \cdot \overline{x_i^1} \\ y_i^0 &= (X_i^3 \oplus x_i^2) \oplus x_i^0 \end{aligned} \tag{B.7}$$

UNIVERSIDAD DE MÁLAGA

- Conversión RBCD$_{\text{-}4421}$ a RBCD

$$\begin{aligned} Y_i^3 &= X_i^3 \cdot \overline{x_i^2} \\ y_i^2 &= X_i^3 \oplus x_i^2 \\ y_i^1 &= x_i^1 \\ y_i^0 &= x_i^0 \end{aligned} \tag{B.8}$$

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Apéndice C
Descomposición para suma Multiformato mediante Diseño Específico

Presentamos un método para construir olDFA$_{Mformat}$ que soporte una de las entradas con RBCD y la otra entrada con cualquier código redundante.

Las funciones presentadas en este apéndice tratan con algunos submódulos del módulo de `descomposición del Mformat` (específicamente, los submódulos BB y AB/BA, donde el código A es RBCD y el código B es otro código redundante estudiado. Hay que tener en cuenta que el submódulo AA corresponde al módulo de descomposición del RBCD y que ha sido presentado en el texto principal). El método para construir los submódulos depende de los bits del código cuyo peso difiere de los bits correspondientes del código RBCD

C.1. Decomposición para X421

Los códigos que difieren solo en el bit más significativo son: -7421, -6421, -5421 y -4421.

Hay que tener en cuenta que hay dos codificaciones que tienen paridad impar, -7421 y -5421, y dos con paridad par, -6421 y -4421.

Usamos la ecuación (C.1) para obtener $T^0_{i+1}, t^0_{i+1}, Z^2_i, z^2_i$ y Z^1_i. Este grupo solo representa los dígitos pares debido al echo de que los pesos de Z_is son pares:

UNIVERSIDAD DE MÁLAGA

$$(\overline{X_i^3} \cdot \overline{Y_i^3}) \cdot (...) + (X_i^3 \oplus Y_i^3) \cdot (...) + (X_i^3 \cdot Y_i^3) \cdot (...) \tag{C.1}$$

Los (...) es una tabla de verdad de tres variables (x_i^2, x_i^1 y y_i^2) con ocho combinaciones en dos niveles lógicos con puertas de tres entradas con el objetivo de prevenir un retardo extra. Hay que observar que los dígitos impares producidos con -7421 y RBCD$_{-5421}$ cuando $X_i^3 \oplus Y_i^3 = 1$ deben ser representados usando el grupo v, añadiendo -1 a v, por lo que obtenemos la siguiente ecuación:

$$\begin{aligned} (X_i^3 + Y_i^3 + x_i^2 + y_i^2 + x_i^1) = \\ = (Z_i^2 + z_i^2 + Z_i^1 + t_{i+1}^0 + T_{i+1}^0) - 1 \\ (y_i^1 + x_i^0 + y_i^0) - 1 = (v_i^2 + v_i^1 + V_i^0) \end{aligned} \tag{C.2}$$

Las ecuaciones del grupo v se obtienen mediante la siguiente tabla de verdad:

Tabla C.1: Tabla de verdad para v

			cv=0			cv=1		
y_i^1	y_i^0	x_i^0	v_i^2	v_i^1	V_i^0	v_i^2	v_i^1	V_i^0
0	0	0	0	0	0	0	0	1
0	0	1	0	0	1	0	0	0
0	1	0	0	1	1	0	0	0
0	1	1	0	1	0	0	1	1
1	0	0	0	1	0	0	1	1
1	0	1	1	0	1	0	1	0
1	1	0	1	0	1	0	1	0
1	1	1	1	0	0	1	0	1

$$\begin{aligned} v_i^2 &= \overline{cv} \cdot (y_i^1 \cdot (y_i^0 + x_i^0)) + y_i^1 \cdot x_i^0 \cdot y_i^0 \\ v_i^1 &= \overline{cv} \cdot (y_i^1 \oplus (y_i^0 + x_i^0)) + cv \cdot (\overline{y_i^1 \oplus (x_i^0 \cdot y_i^0)}) \\ v_i^0 &= cv \oplus (y_i^0 \oplus x_i^0) \end{aligned} \tag{C.3}$$

Destacar que la variable cv es 1 cuando el grupo v recibe un -1 del grupo u, en caso de que la paridad de éste último fuese impar.

Para la descomposición con -7421 y RBCD$_{-5421}$ obtenemos $cv = X_i^3 \oplus Y_i^3$, y para la descomposición de -6421 y -4421, $cv = 0$ debido a que todos los pesos son pares.

UNIVERSIDAD DE MÁLAGA

C.1.1. Descomposición Mixta (dígito RBCD + X421 con peso par)

Para realizar la descomposición mixta para los códigos -6421 y -4421, el grupo v no es modificado y el grupo u $(Z_i^2, z_i^2, Z_i^1, t_i^0, T_i^0)$ cambia cuando $X_i^3 = 1$. Las ecuaciones para cada bit en el grupo u son:

$$\begin{aligned} X_i^3 \cdot \overline{Y_i^3} \cdot (...) + X_i^3 \cdot Y_i^3 \cdot (...) \\ cv_{neg} = 0 \end{aligned} \tag{C.4}$$

$$\begin{aligned} v_{i(neg)}^2 &= cv_{neg} \cdot (y_i^1 \cdot (y_i^0 \oplus x_i^0)) \\ v_{i(neg)}^1 &= cv_{neg} \cdot (y_i^0 + x_i^0) \\ v_{i(neg)}^0 &= cv_{neg} \end{aligned} \tag{C.5}$$

C.1.2. Descomposición Mixta (dígito RBCD + X421 con peso impar)

Para las codificaciones -7421 y -5421, cuando $X_i^3 = 1$ el grupo v cambia debido a que el peso de X_i^3 cambia de par a impar y el valor del grupo u cambia de par a impar (y viceversa). Usamos la Tabla C.1 para resolver este cambio en el grupo v comparando los valores de v_i^2, v_i^1 y V_i^0 entre los casos donde cv es igual a 0 y cuando cv es igual a 1, obteniendo la tabla de verdad mostrada en la Tabla C.2. Además, el grupo u cambia cuando $X_i^3 = 1$.

Tabla C.2: Tabla de verdad para v_{neg}

			cv_{neg}=1		
y_i^1	y_i^0	x_i^0	v_i^2	v_i^1	V_i^0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	0	0	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	0	0	1

UNIVERSIDAD DE MÁLAGA

$$X_i^3 \cdot \overline{Y_i^3} \cdot (...) + X_i^3 \cdot Y_i^3 \cdot (...) \qquad (C.6)$$
$$cv_{neg} = X_i^3$$

C.2. Decomposición para XX21

La descomposición para los códigos -7321, -6321, -6221, -5321, -5221, -4321, -4221 viene dada por la siguiente ecuación:

$$(\overline{X_i^3} \cdot \overline{Y_i^3}) \cdot (...) + (X_i^3 \oplus Y_i^3) \cdot (...)(X_i^3 \cdot Y_i^3) \cdot (...) \qquad (C.7)$$

El grupo v cambia cuando los códigos con X_i^3 y/o x_i^2 tienen peso impar, por ejemplo -7321, -6321, -5321, -4321. Utilizamos la Tabla C.1 y la ecuación (C.4) para obtener el valor correcto de los bits v_i^2, v_i^1 y V_i^0 . Usando estos códigos hay dos casos posibles dependiendo de la paridad de los pesos de los bits.

Para códigos -7321 y -5321:

$$cv = (X_i^3 \oplus x_i^2) \oplus (Y_i^3 \oplus y_i^2)$$

Para códigos -6321 y -4321:

$$cv = (x_i^2 \oplus y_i^2)$$

C.2.1. Descomposición Mixta (dígito RBCD + XX21 ambos con peso impar)

El grupo u de la descomposición con un operando RBCD y otro -7321 o -532 cambia cuando $(X_i^3 + x_i^2) = 1$. Por ello, la expresión de los bits del grupo u es:

$$(\overline{X_i^3} \cdot x_i^2) \cdot (...) + (X_i^3 \cdot \overline{x_i^2}) \cdot (...)(X_i^3 \cdot x_i^2) \cdot (...) \qquad (C.8)$$

donde (...) es una tabla de verdad con tres variables, Y_i^3, x_i^1 y y_i^2.

Para códigos -7321 y -5321, el grupo v cambia cuando cv_{neg}, usado en la ecuación (C.6), es igual a 1, y la expresión de cv_{neg} debe ser:

UNIVERSIDAD DE MÁLAGA

$$cv_{neg} = (X_i^3 \oplus x_i^2)$$

C.2.2. Descomposición Mixta (dígito RBCD + XX21 uno con peso par y otro impar)

El grupo u de la descomposición con un operando RBCD y otro -6321 o -4321 cambia cuando $(X_i^3 + x_i^2) = 1$. Para ello, la expresión de los bits del grupo u es definida en la ecuación (C.8).

Para los códigos -6321 y -4321 el grupo v cambia cuando cv_{neg} es igual a uno. Por ello, la expresión de cv_{neg} debe ser:

$$cv_{neg} = x_i^2$$

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Bibliografía

[1] A. Aswal, M.G. Perumal, and G.N. Srinivasa Prasanna. On Basic Financial Decimal Operations on Binary Machines. *IEEE Transactions on Computers*, 61(8):1084–1096, 2012. (Cited on pages 1 and 5)

[2] A. Avizienis. Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 2, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990. (Cited on page 9)

[3] T. Bjork. *Arbitrage Theory in Continuous Time*. Cambridge Univ. Press, 2004. (Cited on page 61)

[4] M. F. Cowlishaw. Densely Packed Decimal Encoding. In *Proceedings of IEE Proceedings - Computers and Digital Techniques*, volume 149, pages 102–104, May 2002. (Cited on page 2)

[5] M.F. Cowlishaw. Decimal Floating-Point: Algorism for Computers. In *Proceedings of 16th IEEE Symposium on Computer Arithmetic (ARITH)*, pages 104–111, 2003. (Cited on page 1)

[6] A. Y. Duale, M. H. Decker, H.-G. Zipperer, M. Aharoni, and T. J. Bohizic. Decimal Floating-Point in z9: an Implementation and Testing Perspective. *IBM Journal of Research and Development*, 51(1/2), 2007. (Cited on page 2)

[7] L. Eisen, J. W. Ward, H.-W. Tast, N. Mading, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough. IBM POWER6 Accelerators: VMX and DFU. *IBM Journal of Research and Development*, 51(6):1 –21, nov. 2007. (Cited on page 2)

[8] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann, San Francisco, 2004. (Cited on pages 3, 26, 65 and 66)

UNIVERSIDAD DE MÁLAGA

[9] M.D. Ercegovac and T. Lang. On-line Arithmetic for DSP Applications. In *Proceedings of 32nd Midwest Symposium on Circuits and Systems*, pages 365 –368 vol.1, aug 1989. (Cited on page 3)

[10] C. Garcia, S. Gonzalez-Navarro, J. Villalba, and E. Zapata. On-line decimal adder with RBCD representation. In *Proceedings of the IEEE 23rd International Conference on Application-Specific Systems, Architectures, and Processors*, pages 53–60, July 2012. (Cited on page 6)

[11] C. Garcia Vega, S. Gonzalez-Navarro, P. Balboa-La Chica, and J. Villalba. Decimal multiformat online addition. *Computers, IEEE Transactions on*, 65(10):3203–3209, 2016. (Cited on pages 6 and 25)

[12] C. Garcia-Vega, S. Gonzalez-Navarro, J. Villalba, and E. L. Zapata. Decimal online multioperand addition. In *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 350–354, Nov 2012. (Cited on page 6)

[13] B. Girau and A. Tisserand. On-line Arithmetic-based Reprogrammable Hardware Implementation of Multilayer Perceptron Back-Propagation. In *Proceedings of Fifth International Conference on Microelectronics for Neural Networks*, pages 168 –175, feb 1996. (Cited on page 3)

[14] H. H. Goldstine and A. Goldstine. The Electronic Numerical Integrator and Computer (ENIAC). *IEEE Annals of the History of Computing*, 18(1):10–16, 1996. (Cited on page 2)

[15] J. Alexander Gordon, William F. Sharpe, and Jeffery V. Bailey. *Fundamentals of Investments*. Prentice-Hall Int. Ed., 1993. (Cited on page 60)

[16] S. Gorgin and G. Jaberipur. Fully Redundant Decimal Arithmetic. In *Proceedings of 19th IEEE Symposium on Computer Arithmetic (ARITH)*, pages 145–152, June 2009. (Cited on pages 2, 4, 9, 11, 12, 15, 18, 24, 36, 39, 52, 65, 71, 93, 96, 98, 99, 100, 101, 103 and 104)

[17] Liu Han and Seok-Bum Ko. High-Speed Parallel Decimal Multiplication with Redundant Internal Encodings. *IEEE Transactions on Computers*, 62(5):956–968, May 2013. (Cited on page 65)

[18] G. Jaberipur and B. Parhami. Posibits, Negabits, and their Mixed Use in Efficient Realization of Arithmetic Algorithms. In *Proceedings of 15th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*, pages 3 –9, sept. 2010. (Cited on page 11)

UNIVERSIDAD DE MÁLAGA

[19] R.N Kalla et al. Power7: IBM's Next-Generation Server. *IEEE Micro*, 30(2):7 –15, 2010. (Cited on page 2)

[20] T. Lang and A. Nannarelli. A radix-10 digit-recurrence division unit: Algorithm and architecture. *Computers, IEEE Transactions on*, 56(6):727–739, June 2007. (Cited on pages 81, 82 and 88)

[21] R. McIlhenny and M. Ercegovac. On the Design of a Radix-10 Online Floating-Point Multiplier. In *Proceedings of SPIE on Advanced Signal Processing Algorithms, Architectures, and Implementations*, Aug 2009. (Cited on pages 61 and 65)

[22] P. Montuschi and L. Ciminiera. Over-redundant digit sets and the design of digit-by-digit division units. *IEEE Transactions on Computers*, 43(3):269–277, Mar 1994. (Cited on page 81)

[23] J.V. Moreno, T. Lang, and J. Hormigo. Radix-2 Multioperand and Multiformat Streaming Online Addition. *IEEE Transactions on Computers*, 61(6):790–803, June 2012. (Cited on page 17)

[24] J. Moskal, E. Oruklu, and J. Saniie. Design and Synthesis of a Carry-Free Signed-Digit Decimal Adder. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1089 –1092, may 2007. (Cited on page 4)

[25] W.G. Natter and B. Nowrouzian. Digit-serial Online Arithmetic for High-speed Digital Signal Processing Applications. In *Proceedings of Thirty-Fifth Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 171 –176 vol.1, nov. 2001. (Cited on page 3)

[26] H. Nikmehr, B. Phillips, and C. C Lim. A Decimal Carry-free Adder. *SPIE Symposium on Smart Materials, Nano-, and Micro-Smart Systems*, 5649:786–797, February 28 2005. (Cited on page 4)

[27] H. Nikmehr, B. Phillips, and C. C. Lim. Fast decimal floating-point division. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(9):951–961, Sept 2006. (Cited on page 81)

[28] Behrooz Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, Oxford, UK, 2000. (Cited on page 81)

[29] S. Rajagopal and J.R. Cavallaro. Truncated Online Arithmetic with Applications to Communication Systems. *IEEE Transactions on Computers*, 55(10):1240 –12529, oct. 2006. (Cited on page 3)

UNIVERSIDAD DE MÁLAGA

[30] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw. Decimal Floating-Point Support on the IBM System z10 Processor. *IBM Journal of Research and Development*, 53(1):4:1 –4:10, january 2009. (Cited on page 2)

[31] B. Shirazi, D.Y.Y. Yun, and C.N. Zhang. RBCD: redundant binary coded decimal adder. *IEE Proceedings E Computers and Digital Techniques*, 136(2):156–160, March 1989. (Cited on pages v, 9, 11 and 23)

[32] SilMinds. DFP Unit (DFPU). `http://www.silminds.com/ip-products/dfp-unit`, April 2015. (Cited on page 2)

[33] S. Singh, Seung hyun Pan, and M. Ercegovac. Accelerating the Photon Mapping Algorithm and its Hardware Implementation. In *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pages 149 –157, sept. 2011. (Cited on page 3)

[34] A. Svoboda. Decimal Adder with Signed Digit Arithmetic. *IEEE Transactions on Computers*, C-18(3):212–215, 1969. (Cited on page 4)

[35] A. Vazquez, E. Antelo, and J.D. Bruguera. Fast Radix-10 Multiplication Using Redundant BCD Codes. *IEEE Transactions on Computers*, 63(8):1902–1914, Aug 2014. (Cited on pages 65, 76, 77 and 78)

[36] A. Vazquez, E. Antelo, and P. Montuschi. A New Family of High-Performance Parallel Decimal Multipliers. In *Proceedings of 18th IEEE Symposium on Computer Arithmetic (ARITH)*, pages 195–204, June 2007. (Cited on pages 2, 25, 41 and 60)

[37] A. Vazquez, E. Antelo, and P. Montuschi. A radix-10 srt divider based on alternative bcd codings. In *2007 25th International Conference on Computer Design*, pages 280–287, Oct 2007. (Cited on page 81)

[38] A. Vazquez, E. Antelo, and P. Montuschi. Improved Design of High-Performance Parallel Decimal Multipliers. *IEEE Transactions on Computers*, 59(5):679–693, May 2010. (Cited on pages 25 and 60)

[39] A. Vazquez, J. Villalba, and E. Antelo. Computation of Decimal Transcendental Functions Using the CORDIC Algorithm. In *Proceedings of 19th IEEE Symposium on Computer Arithmetic (ARITH)*, pages 179–186, June 2009. (Cited on pages 25, 60 and 61)

[40] A. Vazquez, J. Villalba-Moreno, E. Antelo, and E.L. Zapata. Redundant Floating-Point Decimal CORDIC Algorithm. *IEEE Transactions on Computers*, 61(11):1551–1562, Nov 2012. (Cited on pages 25 and 60)

UNIVERSIDAD DE MÁLAGA

[41] J. Villalba, J. Hormigo, J.M. Prades, and E.L. Zapata. On-line multioperand addition based on on-line full adders. In *Proceedings of IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP)*, pages 322–327, July 2005. (Cited on pages 15, 17 and 19)

[42] Liang-Kai Wang and M. J. Schulte. Decimal floating-point division using newton-raphson iteration. In *Proceedings. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004.*, pages 84–95, Sept 2004. (Cited on page 81)

[43] T. Yoshida, T. Maruyama, Y. Akizuki, R. Kan, N. Kiyota, K. Ikenishi, S. Itou, T. Watahiki, and H. Okano. Sparc64 X: Fujitsu's New-Generation 16-Core Processor for Unix Servers. *IEEE Micro*, 33(6):16–24, Nov 2013. (Cited on page 2)

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA